



AMICA FOR AMS
SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 1/212
File: AmicaSoftware.doc

TITLE	SOFTWARE DESCRIPTION
DOCUMENT TYPE	REPORT
DOC No.	AMST/ SWDS /1/A
ISSUE No.	1
DATE	Jun 07

Prepared by: F. RONCELLA *Fabio Roncella*

Approved by: P. TRAMPUS (CARSO) *Pasolo Trampus*



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 3/212
File: AmicaSoftware.doc

Table of Contents

1. Introduction	5
2. Power-up procedure.....	5
2.1. Bootstrap code.....	5
2.1.1. Loader code check algorithm.....	6
2.1.2. Startup-related memory allocation.....	6
2.2. Loader program.....	7
2.2.1. Main Program loading strategy.....	7
2.2.2. 16-bit CRC algorithm calculation.....	8
2.2.3. Housekeeping data management.....	9
2.2.4. Basic serial communication routines.....	9
2.2.5. Loader data input.....	10
2.2.6. Loader data output.....	12
3. Main Program.....	14
3.1. Program memory usage.....	14
3.2. Clean startup.....	14
3.3. Watchdog.....	14
3.4. Camera parameters and image analysis.....	14
3.4.1. Ram Image Buffers.....	15
3.4.2. Camera Image transfer.....	15
3.5. Stars identification and attitude determination.....	16
3.5.1. Star search.....	16
3.5.2. On-board catalogue.....	17
3.5.3. Attitude calculation strategy.....	17
3.5.4. Reference system for pointing data.....	18
3.5.5. Pointing Data update.....	18
3.6. Synchronization.....	18
3.7. Housekeeping data.....	19
3.8. Internal memory FIFOs and Readout rate.....	19
3.9. Serial communication.....	22
3.10. Commands Interpreter.....	23
3.10.1. Pointing Data request command.....	23
3.10.2. Pointing Data output.....	23
3.10.3. Attitude Quaternion command.....	28
3.10.4. Attitude Roll-Pitch-Yaw command.....	29
3.10.5. Orbit TLE command.....	29
3.10.6. Time command.....	30
3.10.7. Housekeeping data.....	30
4. Simulations.....	32
5. Source Files.....	38
5.1. Bootstrap.....	38
5.1.1. bootstrp.ach.....	38
5.1.2. bootstrp.asm.....	38
5.1.3. def21020.h.....	40
5.1.4. defboot.h.....	43
5.2. Loader.....	44
5.2.1. Loader.ach.....	45
5.2.2. copy.asm.....	47
5.2.3. copy1.asm.....	49
5.2.4. delflash.asm.....	51



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 4/212
File: AmicaSoftware.doc

5.2.5.	utyflash.asm	54
5.2.6.	wait.asm	58
5.2.7.	wriflash.asm	58
5.2.8.	loader.c	63
5.2.9.	def21020.h	84
5.2.10.	defs.h	87
5.3.	Tracker	88
5.3.1.	Interrupt.txt	88
5.3.2.	tracker.ach	89
5.3.3.	delflash.asm	91
5.3.4.	utyflash.asm	94
5.3.5.	wait.asm	98
5.3.6.	wriflash.asm	98
5.3.7.	ImagePro.c	103
5.3.8.	marq.c	121
5.3.9.	nrutil2.c	121
5.3.10.	psfintegal.c	128
5.3.11.	star_find.c	131
5.3.12.	Tracker	166
5.3.13.	tracker1.c	182
5.3.14.	utils.c	188
5.3.15.	def21020.h	190
5.3.16.	defs.h	193
5.3.17.	imagepro.h	195
5.3.18.	nrutil2.h	197
5.3.19.	psfintegral.h	200
5.3.20.	signal.h	200
5.3.21.	star_find.h	208
5.3.22.	utils.h	211

1. INTRODUCTION

AMICA (Astro Mapper for Instrument Check of Attitude) Star Tracker (AST) software collects star fields images from two cameras, processing them to get, after comparison with an on-board astrometric star catalogue, accurate AMS-2 attitude informations. In this way, AMS-2 events will be associated with precise directional data.

First of all, the software selects the camera for the acquisition, acquires an image and determines the thresholds. This operation gives informations on the content of the image, especially about the potential presence of stellar images. A candidate raw direction is available from previous determinations, or is provided by the ISS, or has to be determined by scanning. A target star field is loaded from the catalogue and projected on the CCD image plane. A comparison with the imaged stars is made, looking for a correspondence between the inter-angles of the catalogue stars and the inter-angles of the imaged stars. After recognition, the software transforms the camera reference system into the AMS one. AMS attitude is determined at rates up to 20Hz with an accuracy of a few arcseconds. Amica software will keep in a FIFO a set of directional parameters associated with the internal time of the acquisition of the corresponding image, always ready to deliver a valid data when requested.

Other software functions manage the serial communication with the USCM interface (Universal Slow Control Motion) to exchange the attitude and housekeeping data and commands. It is also possible to upload a revised version of the software or a new star catalogue. The on-board star catalog is derived from the Smithsonian Astrophysical Observatory (SAO) catalog and contains stars up to magnitude 7.

The system is based on ADSP-21020 Digital Signal Processor. Description of software modules is presented.

2. POWER-UP PROCEDURE

2.1. Bootstrap code

At power-up and/or after reset, the ADSP-21020 automatically fetches its first instruction from location 0x08 of the flash program memory; a jump leads to address 0x100, where the **Bootstrap** program execution starts. The Bootstrap code has a length of 256 words, from address 0x100 to 0x1FF of the flash memory; in fact, the first part of the flash memory is reserved for Interrupt servicing (see Table.1 and Table.2).

The Bootstrap program loops over three identical copies of the **Loader** program. These three copies, with the Bootstrap code too, are in the write protected part of the flash memory. In fact, a jumper removal prevents from write operations, so the content of this part of the memory cannot be modified by a software operation. The write-protected part of the flash memory is in its first 32 KWords, from 0x000000 to 0x007FFF address.

The remaining part of the flash memory, for a total of 4 MWords, from 0x008000 to 0x3FFFFFF, accepts read/write operations. Several copies (three) of the main program, and also the star Catalogue (two copies), are stored in this writable part of the flash memory.

2.1.1. Loader code check algorithm

The **Bootstrap** code performs a majority check of the three (presumed) identical copies of the Loader program. At the same time, the result (modal value) of each majority check operation is written at the beginning of the Program Memory Ram, starting at address 0x400000.

The error correction is made bit-to-bit, based on a majority check over the three copies of Loader program stored in flash memory. In detail, each 48-bit instruction of each copy of the Loader is loaded in PX register; 16 bits in PX1, the remaining 32 bits in PX2. The majority comparison is applied separately to the 16-bit triplet and then to the 32-bit triplet, with the re-use of the same registers. Naming the elements of each triplet with p_1 , p_2 and p_3 , the majority element p_{maj} is calculated as follows:

$$p_{maj} = ((p_1 \text{ xor } p_2) \text{ and } (p_3)) \text{ or } ((\text{not}(p_1 \text{ xor } p_2)) \text{ and } (p_1))$$

There is a unique counter for the total number of the majority exception found. The counter value is the number of PX1 or PX2 registers found with a wrong value (1 or more bits, typically 1) that has been corrected by the majority algorithm. An estimate of the number of errors found in each of the three copies of the Loader program is possible but onerous compared to the real benefits (in fact the computational charge would be doubled).

At the end of the process, the Loader program just built in PM Ram is automatically executed, starting at address 0x400000.

2.1.2. Startup-related memory allocation

Table.1 shows the allocation of the Flash Program Memory address space; *Table.2* shows the allocation of the Ram Program Memory address space; *Table.3* shows types, addresses, length and size of Program Memory and Data Memory.

Address range	Length (dec)	Length (hex)	Usage	Comments
0x000000-0x000007	8	8	reserved	HW write protected
0x000008-0x0000FF	248	F8	Interrupts servicing	HW write protected
0x000100-0x0001FF	256	100	Bootstrap code	HW write protected
0x000200-0x0002FF	256	100	Loader runtime header, #1	HW write protected
0x000300-0x0022FF	8.192	2000	Loader code, #1	
0x002300-0x0023FF	256	100	Loader initializations, #1	
0x002400-0x0024FF	256	100	Loader runtime header, #2	HW write protected
0x002500-0x0044FF	8.192	2000	Loader code, #2	
0x004500-0x0045FF	256	100	Loader initializations, #2	
0x004600-0x0046FF	256	100	Loader runtime header, #3	HW write protected
0x004700-0x0066FF	8.192	2000	Loader code, #3	
0x006700-0x0067FF	256	100	Loader initializations, #3	
0x006800-0x007FFF	6.144	1800	spare	HW write protected
0x008000-0x00FFFF	32.768	8000	Tracker v1 validation	not protected
0x010000-0x03FFFF	196.608	30000	Tracker v1	not protected
0x040000-0x047FFF	32.768	8000	Tracker v2 validation	not protected
0x048000-0x077FFF	196.608	30000	Tracker v2	not protected

0x078000-0x07FFFF	32.768	8000	Tracker v3 validation	not protected
0x080000-0x0AFFFF	196.608	30000	Tracker v3	not protected
0x0B0000-0x16FFFF	786.432	C0000	Star Catalogue 1	not protected
0x170000-0x22FFFF	786.432	C0000	Star Catalogue 2	not protected
0x230000-0x3FFFFF	1.900.544	1D0000	spare	not protected

Table.1: Allocation of the PM FLASH address space.

Address range	Length (dec)	Length (hex)	Usage	Comments
0x400000-0x4000FF	256	100	Loader runtime header	majority check result
0x400100-0x4020FF	8.192	2000	LOADER	
0x402100-0x4021FF	256	100	Loader initializations	
0x402200-0x402200	1	1	counter	majority exceptions
0x402201-0x406FFF	19.967	4DFF	spare	
0x407000-0x4070FF	256	100	Tracker runtime header	
0x407100-0x436EFF	196.096	2FE00	Tracker Main Program	loaded from Flash
0x436F00-0x436FFF	256	100	Tracker initializations	
0x437000-0x43FFFF	36.864	9000	spare	

Table.2: Allocation of the PM RAM address space.

	Type	Addresses	Length	Size (bits)	Notes
PM	program flash	0x000000-0x3FFFFFF	4Mword	48	R (W)
PM	program ram	0x400000-0x43FFFF	256Kword	48	R/W
DM	data ram	0x000000-0x03FFFF	256Kword	40	R/W
DM	data ram	0x040000-0x07FFFF	256Kword	16	R/W
DM	image buffer	0x080000-0x0BFFFF	256Kword	16	FPGA
DM	serial port 1	0xC0000-0xC0003	4	8	FPGA
DM	serial port 2	0xC0004-0xC0007	4	8	FPGA
DM	FPGA register	0xC0008-0xC0008	1	16	FPGA
DM	Timer register	0xC0009-0xC0009	1	32	FPGA
DM	Camera register	0xC000A-0xC000A	1	16	FPGA

Table.3: Types, addresses, length and size of Program Memory and Data Memory.

2.2. Loader program

The **Loader** program, small and robust, implements a basic set of functions:

- Main program loading strategy;
- 16-bit CRC algorithm calculation;
- Housekeeping data management;
- Basic serial communication routines.

2.2.1. Main Program loading strategy

The Loader manages the loading of the **Star Tracker** main program. There will be three copies of it, stored in the writable part of the flash memory, i.e. the most: from 0x008000 to 0x0AFFFF. Each copy will



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 8/212
File: AmicaSoftware.doc

be marked as “valid” and/or “present” by its first sector of memory, 32 KWords wide. If this sector has a value different from 0xFFFF... it means that the corresponding copy is marked as valid, otherwise it cannot be used, because the program is not present, or it is present but it has been marked as non-valid, e.g. because it is still under test.

If the Loader, looking for the latest version first, finds a program with this initial validation, it calculates its Frame Check Sequence (FCS) using a 16-bit CRC algorithm, and then compares its result with the FCS value stored at the end of the same program. If this final validation is achieved, and there isn't an external request received from the serial port, the Loader can load and execute the program.

During **Bootstrap** program operation, the interrupts generated by the serial ports are not serviced. In fact, when enabled, these interrupts have to be serviced almost immediately, because each byte received from the serial channel is overwritten by the following one. Once the good version of the **Loader** program is running, the serial communication is performed again without servicing the interrupts from the serial ports; in fact, the program loop is fast enough to manage characters coming from both serial ports without losing any of them. The serial communication interrupts starts to be serviced only when the **Main Program** is executed; this choice leads to a great simplification in a critical task such the interrupts management, avoiding the adoption of a conditional multiple jump mechanism.

After the boot, the Loader program waits a predefined time (e.g. 10 seconds), looking for a command from the serial port. If this time elapses without receiving commands, the Loader executes the described procedure to load the default (latest) Main Program, if any; if there is no valid program to load, the Loader remains in an infinite loop, servicing commands from the serial ports. Otherwise, if a command is received, the Loader executes it, depending on the command type.

In particular, if the command is asking for the execution of a specified program, the Loader performs the usual CRC check over it and, if validated, loads it and finally passes the control to its first instruction; if the command is of a different type (e.g. write flash, read flash, read housekeeping data), it is serviced, and the Loader remains in an infinite loop, servicing subsequent commands until a “reset” command or a “load program N” command is received. The procedure is described in the diagram in *Figure.1*.

As stated, during Loader program operation the interrupts servicing is still disabled, so the initial standby Timer is implemented simply using a counter in the program main loop, calibrated on the processor typical cycle time.

2.2.2. 16-bit CRC algorithm calculation

The 16-bit CRC algorithm implemented in the Loader code uses the 0x1021 polynomial (i.e. $g = x^{16} + x^{12} + x^5 + x^0$):

```
unsigned long crc_table[256];
int crc_table_computed = 0;

/* Build a table for a fast CRC calculation */
void make_crc_table(void)
{
    unsigned long c;
    int n, k;

    for(n = 0; n < 256; n++)
    {
        c = (unsigned long) n;
        for(k = 0; k < 8; k++)
        {
            if(c & 1)
                c = 0xedb88320L ^ (c >> 1);
            else
                c = c >> 1;
        }
    }
}
```

```
                c = c >> 1;
            }
            crc_table[n] = c;
        }
        crc_table_computed = 1;
    }

/* Update a running CRC with the bytes buf[0..len-1]. */
/* The CRC should be initialized to all 1's, and the transmitted */
/* value is the 1's complement of the final running CRC. */

unsigned long update_crc(unsigned long crc, unsigned char *buf, int len)
{
    unsigned long c = crc;
    int n;

    if(!crc_table_computed)
        make_crc_table();
    for(n = 0; n < len; n++)
    {
        c = crc_table[(c ^ buf[n]) & 0xff] ^ (c >> 8);
    }
    return c;
}

/* Return the CRC of the bytes buf[0..len-1]. */
unsigned long crc(unsigned char *buf, int len)
{
    return update_crc(0xffffffffL, buf, len) ^ 0xffffffffL;
}
```

2.2.3. Housekeeping data management

The Loader answer to a “Get Housekeeping” command returns 16 raw values, read from FPGA; they are 12-bit values (power supply parameters and onboard temperature sensors) from ADCs, stored in 16-bit words. These raw values have to be post-processed externally to get calibrated data, if needed. On the contrary, the main Star Tracker program performs these calculations internally, and the available housekeeping data set will be considerably larger, with informations about cameras (that are still switched off during boot) and the main program status itself.

Normally, during Main Program operation, the housekeeping values are read using an interrupt, signaling a “data ready” status. During Loader operation, on the contrary, these values can be read directly and asynchronously; in fact, the interrupts management is momentary disabled, to avoid, again, the adoption of a conditional multiple jump mechanism. This strategy is robust enough, because the housekeeping values are updated at a low rate (about 20 times per second), so they could be read in the middle of a write operation only rarely. Anyway, the Loader will always read them twice, comparing the values; if they are different they are read again immediately; at this point the values will be definitely valid, because this operation is much faster than housekeeping values update itself.

2.2.4. Basic serial communication routines

After initialization of DSP registers, the Loader looks for the presence of commands; a basic set of command is implemented.

Serial ports status registers are checked for the presence of new characters. If a new character is found, it is immediately added to a buffer. In each program loop, each buffer (one per serial port) is checked for the presence of a complete command; if a command is present, it is sent to a command queue, to be interpreted and executed. The message CRC is checked, to validate the message itself; at this point, the command identifier is checked and the proper action is taken.

When a message has to be transmitted, first it is built in a buffer, fed with the proper data content; then it is sent character by character, using the selected serial port.

The basic set of commands implemented in Loader program allows read/write operations on flash memory and ram memory, readout of housekeeping data (e.g. for diagnostic purposes) and selection of a specific program to be executed.

2.2.5. Loader data input

The structure of a generic command accepted by Loader program is described below:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	number of bytes in message
2	Command ID	word 0	
3	Command ID	word 1	
4	<i>data 0</i>	<i>byte</i>	
...	
<i>NB-3</i>	<i>data NB-7</i>	<i>byte</i>	
NB-2	CRC	word 0	
NB-1	CRC	word 1	

Generic Command structure

The first byte contains the sync 0xE5, the second byte has the number of command bytes, the 2nd and 3rd bytes contain the command identifier. The last two bytes contain the CRC of the bytes 0..NB-3. This command structure is the same of the main Star Tracker program; the list of commands accepted by the boot software is a subset of the complete commands set implemented in the main program.

Boot Command

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	7
2	Command ID	word	0x01
3	Command ID	word	0x00
4	<i>Program Version</i>	<i>byte</i>	
5	CRC	word	lo
6	CRC	word	hi

Boot Command contains the index of the main program copy that has to be executed. The Loader performs the usual CRC check over it and, if validated, loads it and passes the control to its first instruction.

Write Command

byte	description	type	notes
0	sync	byte	0xE5

1	message nr_bytes	byte	NB
2	Command ID	word	0x02
3	Command ID	word	0x00
4	addr	word24	0
5	addr	word24	1
6	addr	word24	2
7	length	word	lo
8	length	word	hi
9	data 0	byte	
...	
NB-3	data NB-12	byte	
NB-2	CRC	word	lo
NB-1	CRC	word	hi

Write Command contains the 24 bit starting address, the length of the data segment to be written to Flash memory or Ram memory, and the data to be written.

This command can be used to upload segments of a new program version; when the upload has been completed, and the new program has been validated by writing the corresponding dedicated sector, after a system reboot the new program can be executed.

Read Command

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	11
2	Command ID	word	0x03
3	Command ID	word	0x00
4	addr	word24	0
5	addr	word24	1
6	addr	word24	2
7	length	word	lo
8	length	word	hi
9	CRC	word	lo
10	CRC	word	hi

Read Command contains the starting address of the data segment to read from flash memory or Ram memory and its length.

Get HK Command

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	6
2	Command ID	word	0x04
3	Command ID	word	0x00
4	CRC	word	lo
5	CRC	word	hi

Get HK Data Command is useful to get housekeeping data (temperatures, voltages, currents) for diagnostic purposes. The loader services the command sending a standard set of telemetry values.



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 12/212
File: AmicaSoftware.doc

Reset Command

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	6
2	Command ID	word	0x05
3	Command ID	word	0x00
4	CRC	word	lo
5	CRC	word	hi

Reset Command has no parameters. When issued, the system is rebooted.

2.2.6. Loader data output

The structure of a generic data output generated by Loader program is described below:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	number of bytes in message
2	Message ID	word	lo
3	Message ID	word	hi
4	<i>data 0</i>	<i>byte</i>	
...	
<i>NB-3</i>	<i>data NB-7</i>	<i>byte</i>	
NB-2	CRC	word	lo
NB-1	CRC	word	hi

Generic Data Output

The first byte contains the sync 0xE5, the second byte has the number of message bytes, the 2nd and 3rd bytes contain the message identifier. The last two bytes contain the CRC of the bytes 0..NB-3. Again, this output message structure is the same of the main Star Tracker program.

Answer to Read Command

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	NB
2	Message ID	word	0x03
3	Message ID	word	0x00
4	<i>data 0</i>	<i>byte</i>	
...	
<i>NB-3</i>	<i>data NB-7</i>	<i>byte</i>	
NB-2	CRC	word	lo
NB-1	CRC	word	hi

The Loader answer to a “Read” command returns the requested data segment; optionally, the complete content of the corresponding command, namely the starting address and the length of the data segment, can be returned.

*Answer to **Get HK** Command*

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	38
2	Message ID	word	0x04
3	Message ID	word	0x00
4	hk01	word	12 bit ADC value on 16 bit (0)
5	hk01	word	12 bit ADC value on 16 bit (1)
6	hk02	word	...
7	hk02	word	
8	hk03	word	
9	hk03	word	
10	hk04	word	
11	hk04	word	
12	hk05	word	
13	hk05	word	
14	hk06	word	
15	hk06	word	
16	hk07	word	
17	hk07	word	
18	hk08	word	
19	hk08	word	
20	hk09	word	
21	hk09	word	
22	hk10	word	
23	hk10	word	
24	hk11	word	
25	hk11	word	
26	hk12	word	
27	hk12	word	
28	hk13	word	
29	hk13	word	
30	hk14	word	
31	hk14	word	
32	hk15	word	
33	hk15	word	...
34	hk16	word	12 bit ADC value on 16 bit (0)
35	hk16	word	12 bit ADC value on 16 bit (1)
36	CRC	word	lo
37	CRC	word	hi

The Loader answer to a “Get Housekeeping” command returns 16 raw values; they are 12-bit values from ADCs, stored in 16-bit words. These raw values have to be post-processed externally to get calibrated data, if needed.

The answer to a command without expected returned values, namely “**Boot**”, “**Write**” and “**Reset**” commands, will be whether an “Ok” message or an “Error” message, enclosing the corresponding Command ID.



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 14/212
File: AmicaSoftware.doc

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	7
2	Message ID	word	0x01, 0x02, 0x05
3	Message ID	word	0x00
4	result code	byte	0x00=Ok, 0xFF=Error
5	CRC	word	lo
6	CRC	word	hi

The eventual non-zero returned value points out an unsuccessful result of the corresponding command. This value is also an error code, useful for diagnostic purposes. Obviously, the answer to a "Reset" command, if there isn't an error condition, is issued before its execution.

3. MAIN PROGRAM

After boot operations, Star Tracker main program, copied from the FLASH to the RAM memory in order to increase execution speed, is executed. The scheme in *Figure.2* outlines the main process and the auxiliary processes described in following sections.

3.1. Program memory usage

The scheme in *Figure.3* shows the conceptual organization of flash Program Memory and Ram Program Memory. The picture shows also the programs loading from Flash memory to Ram memory; the bootstrap code loads the Loader program to Ram memory; once executed, the Loader program loads to Ram memory the right version of the Main Program.

3.2. Clean startup

After startup, main program performs a series of initializations. Some of them are performed to avoid potential problems; e.g., after a reset operation a camera could remain powered. So registers are reset, cameras are switched off resetting camera flags FLAG0 and FLAG1, serial and housekeeping registers are cleaned. Usually a processor reset automatically clears the IMASK register, blocking any interrupts from interfering with instructions execution; anyway, at boot interrupts are disabled (bit 12 in MODE1 register), reset, properly masked (IMASK register) and then enabled again.

3.3. Watchdog

FLAG3 on ADSP-21020, once having been set as an output, works as a Watchdog. Normally, in the main loop the program switches it alternatively on and off continuously; in case of problems, after 20 seconds of inactivity the FPGA will perform a RESET.

3.4. Camera parameters and image analysis

The star searching algorithms are calibrated on characteristic parameters of AST cameras, summarized below:



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 15/212
File: AmicaSoftware.doc

Lens Aperture	60 mm
Focal Length	75 mm
f-Ratio	f/1.25
Image Scale at Detector	0.36 $\mu\text{m}/\text{arcsec}$
Field of View	6.3 x 6.3 deg
Pixel Size	44 x 44 arcsec
CCD pixel size	16x16 μm
CCD Image Area Size	8.19 x 8.19 mm
CCD sensitive area	512x512 pixels
A/D Conversion	12 bit
Readout Time	33 ms
Readout Frequency	30 Hz
Readout Noise	22 $\mu\text{V}/\text{pixel}$ (3.2 e^-/pixel) @10°C

Due to such parameters, particularly the sky-per-pixel value, a small defocusing of the image is useful to spread each star image on more pixels, allowing a better centroiding calculation.

3.4.1. Ram Image Buffers

There are two image memories of 256Kword x 16bit. Depending on ADSP-21020 FLAG2, the FPGA connects one of them to the camera interface (camera receivers) and the other one to ADSP-21020. In this way, when the camera interface is writing an image to a memory buffer, the ADSP-21020 can access the other memory buffer. Camera 0 and 1 are turned on by setting FLAG0 and FLAG1, respectively.

3.4.2. Camera Image transfer

As an image arrives from the camera, the DSP receives an interrupt (IRQ2) and the BDS (*buffer destination switch*) register is set to 0 (“no buffer selected”) by the FPGA. When the DSP needs a new image, FLAG2 is set to 0 or 1 to choose the desired Ram Image Buffer (0 or 1); correspondently, the BDS register is set to 1 or 2, respectively. The camera is selected through the 1 bit CSS - *camera selection switch* - Register.

511x511 pixels out of 512x512 are transferred to the processor board. The 512th pixel of each row contains the mask pixel of the row. The 512th row is skipped and replaced by the housekeeping data (temperatures, voltages, status and error conditions). So the image useful for star fields search and identification is 511x511 pixels wide.

Camera housekeeping data includes also informations on minimum, maximum and mean pixel value in current image. A non-zero minimum value is always expected, also after a short exposure, due at least to background noise; a zero value will be regarded as a parity error found during image transfer. The mean pixel value is useful to predict the information content of the image. The mean value is referred to most significant bits, with 4096 levels (from 0 to 4095). A 4095 value means that saturation has occurred, e.g. due to blooming. Values over about 1000 mean that there is “something” in the image; a value like that could be due, for example, to a “difficult” star field, with ten faint stars around 7th magnitude.

These informations are used by the software to understand the current situation, to automatically tune star finding algorithms thresholds, and also to adjust the gain value for next acquisition.

3.5. Stars identification and attitude determination

The attitude of the Star Tracker, characterized by the rotation of its orthogonal cartesian axes with respect to an inertial reference system, has a known relationship with the attitude of the ISS. We refer to the rigid body rotation matrix that relates the ISS body system to an inertial system; this matrix is expressed in terms of the Euler parameters (quaternions) and is initialized to the estimated direction of the Star Tracker, if available.

The identification of the star fields imaged by the Star Tracker is made comparing these stars with those in the on-board catalogue. Before comparison, the stars in the catalogue are projected on the plane of the CCD imager, using the current value of the rotation matrix, i.e. the estimated direction of the Star Tracker. The projection procedure makes use of the co-linearity formulas, which transform celestial coordinates of stars (right ascension and declination) into metric coordinates on the CCD (x and y).

The program searches a correspondence between measured inter-angles, i.e. angles under which star pairs are seen by the Star Tracker, and the inter-angles calculated from the stars in the catalogue. In this way, two inter-angles tables are constructed: one for the imaged stars and one for the catalogue stars. An hypothesis of correspondence between the stars listed in the two tables is made; it will be accepted or rejected by the orientation algorithm. In case of rejection, another hypothesis is built, proceeding in the same way. The algorithm minimizes the distance between imaged stars and the hypothetically corresponding ones from the catalogue. The minimization procedure is made by adjusting the elements of the rotation matrix.

Once a good overlap between imaged star field and catalogue star field is achieved, the latest parameters values in the rotation matrix are used to calculate the orientation of the Star Tracker with respect to the inertial reference system.

A maximum of five stars is used in the processing; a higher number increases calculation time without advantages in results accuracy; with less than three stars the algorithm fails.

3.5.1. Star search

The star finding algorithm involves the analysis of the CCD image to detect and locate positive brightness enhancements; true stellar images should be discriminated from noise peaks, images of extended objects, bad pixels, electric charges due to cosmic rays, rows due to saturation, and so on. Two alternate algorithms have been integrated in the analysis software.

The first algorithm analyzes each pixel trying to numerically fit an analytic Gaussian profile to the brightness values found in the image, in a square around the pixel. The fit, performed using a simplified version of Marquardt solution of a least squares problem, will be very good for a true stellar image, and on the other hand it will be very poor in case of some artifact in the image. The good performance of the algorithm is based on a previous knowledge of the image content, i.e. the expected readout noise of the CCD, the efficiency of the detector (photoelectrons per ADU, analog-to-digital converter units), the maximum brightness level for which the detector still works linearly, the rough value of FWHM (full width at half maximum) of a stellar image in our optical configuration, and the minimum expected pixel level due only to random noise. To get a precise centroiding, each pixel is seen by the algorithm as a matrix of NxN subpixel. The star finding procedure uses a starting set of thresholds calculated on the basis of minimum, maximum and mean value in the image, found in the housekeeping data row of the image itself; if needed, more than one iteration is performed with a suitable modification strategy of the set of thresholds, to find a reasonable number of stars in the imaged field. A second, simpler, algorithm has been integrated in the finding procedure; this one is based on a modified center of gravity calculation performed on pixel values. This procedure is faster, but also less robust, with a lower capability of false stellar image discrimination.

The simulations performed on artificial images (see chapter 4) have led to good results of both algorithms. The software is able to automatically select the suitable algorithm on the basis of the results of previous attitude determinations; if needed, the first algorithm is loaded, with better and safer results, but with a slower execution time.

3.5.2. On-board catalogue

The on-board star catalog is a subset of the Smithsonian Astrophysical Observatory (SAO) catalog and contains stars up to magnitude 7. The parameters of each star are recorded in 20 bytes; double precision variables are used for RA and DEC coordinates, a floating point variable is used for magnitudes.

The catalog has been organized dividing the celestial sphere in sectors, to improve access efficiency to star fields around the current pointing direction. These sectors, with a roughly square shape, are partially overlapping. This leads to a redundancy in catalogue data, with an increase of needed storage in flash memory; but there is an important advantage in the archive access algorithm efficiency, especially during Tracking operation.

In flash memory the storage reserved for star catalogues is enough for about 470000 stars. There will be two alternate versions of the working catalogue, and each of them contains at least 100000 stars, keeping into account the redundant storage. As an option, the searching efficiency of the access algorithm can be increased reducing the size of the archive; limiting its content to stars brighter than 7th magnitude (15945 stars), in 99.4% of times the searching algorithm is able to find at least three stars in the pointed field of view, allowing a successful pointing direction determination. With a larger set of stars the expected performance is still better.

The sky portion extracted from the catalogue by searching algorithm is properly sized keeping into account the field of view of the camera, which is quite large; in fact, its 6.3 x 6.3 degrees FoV covers about 0.87% of the celestial sphere.

The imaged star field is compared with a star field built by projecting the stars from the catalogue on the plane of the CCD, i.e. the plane tangent to the celestial sphere in the current (candidate) pointing direction. As the polar zones are a singularity, an auxiliary catalogue is used, with celestial coordinates rotated by 90° around the axis normal to the plane identified by the celestial polar axis and the vernal equinox.

3.5.3. Attitude calculation strategy

The software implements three image analysis different levels:

- *Full Search*
- *Local Search*
- *Tracking*

Full Search operation is based on a search on the all-sky star atlas; comparing the star field found in the image with the stars in the catalogue, considering relative distances, angles and brightness, the field is recognized and the direction pointed by the camera is retrieved. This analysis level is the slowest, and it is needed when there is no previous knowledge of a candidate, although rough, pointing direction, e.g. after a reboot of the system, and with no ISS attitude informations available.

Local Search is based on the same algorithms, but the analysis is limited to a local star atlas, around raw coordinates available as starting point. In this way the analysis is faster, and the directional information is ready in a shorter time.

When a star field has been recognized and a directional information is available, the software switches its operation to a *Tracking* procedure. Once recognized, a limited number (selectable) of bright star objects is

followed-up through the image field, considering a square of pixels centered on each of them. In this way the directional information is updated continuously, with a fast processing. In this algorithm, when an object approaches the border of the Field-of-View a new one is selected, ready to replace it in the calculation. In the same way, a star catalogue monitoring predicts new objects entering the Field-of-View, ready to be taken into account by the tracking algorithm.

In each image analysis level (see *Figure.4*), a number of consecutive errors (task failed) send back to the upper level. The number of admitted errors, selectable, can be different for each level; N=0 means that we go back to the upper level at the first error. Over the upper level (*Full Search*) a system Reset is performed.

3.5.4. Reference system for pointing data

As seen, pointing data is calculated with respect to a reference system in which relative orientations of ISS, AMS and AST cameras are considered.

The ISS Coordinate System is defined with the positive x-axis pointing along the axis of the U.S. Lab away from the Zarya and Service Modules; the positive y-axis points starboard along the main truss, and positive z completes a right-handed coordinate system.

The AMS reference system derives from the ISS reference system through three successive rotations: YAW=-90°, PITCH=-12°, ROLL=180°.

The ASTC0 (1st AST camera) reference system derives from the AMS one after three successive rotations: YAW=235°, PITCH=0°, ROLL=-40°. The ASTC1 (2nd AST camera) reference system derives from the AMS one after three successive rotations: YAW=55°, PITCH=0°, ROLL=-40°.

3.5.5. Pointing Data update

Considering the 4°/min rotation at zenith due to the 90 minutes ISS orbit, in 1/15 second there is a variation of 16 arc-seconds in the pointing direction. Considering the resolution of the AMICA detector, about 4 arc-seconds, any readout faster than 60 Hz produces consecutive readings not significantly different. In addition, the AMS orientation can be interpolated offline, to determine what it was at a given time; we only need enough readouts to make interpolated results accurate enough. Internally, AMICA software will update the pointing direction with a 15-20 Hz rate. Externally, the USCM will never request orientation data more than once per second, to avoid an excessive load over a communication channel with a limited bandwidth; in fact, the maximum possible data rate between AMICA and USCM is about 750 bytes/second.

3.6. Synchronization

Each specific AMS event must be associated with an accurate pointing direction, provided by Amica. The Star Tracker has an internal clock, built using a simple 32-bit counter of pulses coming from a fixed frequency oscillator (1 KHz). Pointing data sent to the USCM is time-tagged with the internal clock value of the instant of the measurement. When asked for, the Star Tracker will return the most recent measurement available. This operation is asynchronous with the trigger and DAQ system. Also the trigger has an internal clock, again based on a counter of pulses from a fixed frequency oscillator. In the same way, each event is time-tagged with this clock value.

Both systems periodically receive directly from an external source, not via the USCM, a synchronization pulse (LVDS); this pulse produces a reset of the internal clock counter, starting a new "epoch". After the digital sync reset is given, AMS provides an absolute time stamp through the serial line; this absolute time is defined as the time elapsed from 00:00:00 of Jan 01, 2003 (Julian Date = 2452640.5).

In this way, during offline data processing each event will be associated with a pointing data set with the needed accuracy.

3.7. Housekeeping data

FPGA collects system housekeeping data, i.e. functional informations, power supply parameters and onboard temperature sensors readouts; each time a data set is ready, ADSP-21020 receives an interrupt (IRQ0). A complete set of ADCs housekeeping data is prepared and made ready for DSP readout 20 times per second.

The data set contains for each camera (ASTC0 and ASTC1) two temperatures, two current, two voltages, two status words, the working time, the readout value of the four photodiodes; for electronics (ASTE) there are four hardware status words, two temperatures, one voltage and four software status words.

3.8. Internal memory FIFOs and Readout rate

Amica has two dedicated FIFOs in which the latest values of pointing and housekeeping data are stored by the program.

The pointing data FIFO has a length of 12 Kbytes, suitable for 200 measures, each of them is 60 bytes long. Each measure has the following format:

byte	description	type	notes
0	element number	byte	0..199
1	time from sync ms	long 0	
2	time from sync ms	long 1	
3	time from sync ms	long 2	
4	time from sync ms	long 3	
5	abs time s	long 0	seconds elapsed from 1/1/2003
6	abs time s	long 1	(JD-2452640.5)*3600*24
7	abs time s	long 2	
8	abs time s	long 3	
9	abs time 4 ms	byte	fraction in 1/256 s
10	ASTC (0 or 1)	byte	0=starboard, 1=port direction
11	camera z RA	word24 0	1/10 of arcsec 0..360
12	camera z RA	word24 1	
13	camera z RA	word24 2	
14	camera z DEC	word24 0	
15	camera z DEC	word24 1	
16	camera z DEC	word24 2	
17	camera x RA	word24 0	
18	camera x RA	word24 1	
19	camera x RA	word24 2	
20	camera x DEC	word24 0	
21	camera x DEC	word24 1	
22	camera x DEC	word24 2	
23	camera y RA	word24 0	
24	camera y RA	word24 1	
25	camera y RA	word24 2	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 20/212
 File: AmicaSoftware.doc

26	camera y DEC	word24 0	
27	camera y DEC	word24 1	
28	camera y DEC	word24 2	
29	nr of stars in FOV	byte	
30	background	byte 0	measured background/10
31	dark	byte 1	
32	temperature	byte	+/-127 °C
33	star x 0	word 0	in 1/100 of pixel
34	star x 0	word 1	
35	star y 0	word 0	
36	star y 0	word 1	
37	star mgv 0	byte	INT(Log ₂ (intensity)*1000)
38	star x 1	word 0	
39	star x 1	word 1	
40	star y 1	word 0	
41	star y 1	word 1	
42	star mgv 1	byte	
43	star x 2	word 0	
44	star x 2	word 1	
45	star y 2	word 0	
46	star y 2	word 1	
47	star mgv 2	byte	
48	star x 3	word 0	
49	star x 3	word 1	
50	star y 3	word 0	
51	star y 3	word 1	
52	star mgv 3	byte	
53	star x 4	word 0	
54	star x 4	word 1	
55	star y 4	word 0	
56	star y 4	word 1	
57	star mgv 4	byte	
58	recognition	word 0	
59	recognition	word 1	

Pointing FIFO element

With 200 elements for the FIFO, at the maximum acquisition frequency (30 Hz) the last 6.67 seconds of pointing data are stored in memory.

The housekeeping data FIFO has a length of 3200 bytes, suitable for 40 measures, each of them is 80 bytes long. Each measure has the following format:

byte	description	type	notes
0	element number	byte	0..39
1	abs time s	long 0	seconds elapsed from 1/1/2003
2	abs time s	long 1	(JD-2452640.5)*3600*24
3	abs time s	long 2	
4	abs time s	long 3	
5	abs time 4 ms	byte	fraction in 1/256 s



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 21/212
 File: AmicaSoftware.doc

6	temperature1 astc0	word 0	
7	temperature1 astc0	word 1	
8	temperature2 astc0	word 0	
9	temperature2 astc0	word 1	
10	current1 astc0	word 0	
11	current1 astc0	word 1	
12	current2 astc0	word 0	
13	current2 astc0	word 1	
14	voltage1 astc0	word 0	
15	voltage1 astc0	word 1	
16	voltage2 astc0	word 0	
17	voltage2 astc0	word 1	
18	status1 astc0	word 0	
19	status1 astc0	word 1	
20	status2 astc0	word 0	
21	status2 astc0	word 1	
22	time1 astc0	word 0	
23	time1 astc0	word 1	
24	phot1 astc0	word 0	
25	phot1 astc0	word 1	
26	phot2 astc0	word 0	
27	phot2 astc0	word 1	
28	phot3 astc0	word 0	
29	phot3 astc0	word 1	
30	phot4 astc0	word 0	
31	phot4 astc0	word 1	
32	temperature1 astc1	word 0	
33	temperature1 astc1	word 1	
34	temperature2 astc1	word 0	
35	temperature2 astc1	word 1	
36	current1 astc1	word 0	
37	current1 astc1	word 1	
38	current2 astc1	word 0	
39	current2 astc1	word 1	
40	voltage1 astc1	word 0	
41	voltage1 astc1	word 1	
42	voltage2 astc1	word 0	
43	voltage2 astc1	word 1	
44	status1 astc1	word 0	
45	status1 astc1	word 1	
46	status2 astc1	word 0	
47	status2 astc1	word 1	
48	time1 astc1	word 0	
49	time1 astc1	word 1	
50	phot1 astc1	word 0	
51	phot1 astc1	word 1	
52	phot2 astc1	word 0	
53	phot2 astc1	word 1	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 22/212
 File: AmicaSoftware.doc

54	phot3_astc1	word 0	
55	phot3_astc1	word 1	
56	phot4_astc1	word 0	
57	phot4_astc1	word 1	
58	status1_e_aste	word 0	
59	status1_e_aste	word 1	
60	status2_e_aste	word 0	
61	status2_e_aste	word 1	
62	status3_e_aste	word 0	
63	status3_e_aste	word 1	
64	status4_e_aste	word 0	
65	status4_e_aste	word 1	
66	temperature1_aste	word 0	
67	temperature1_aste	word 1	
68	temperature2_aste	word 0	
69	temperature2_aste	word 1	
70	voltage_aste	word 0	
71	voltage_aste	word 1	
72	status1_sw_aste	word 0	
73	status1_sw_aste	word 1	
74	status2_sw_aste	word 0	
75	status2_sw_aste	word 1	
76	status3_sw_aste	word 0	
77	status3_sw_aste	word 1	
78	status4_sw_aste	word 0	
79	status4_sw_aste	word 1	

Housekeeping FIFO element

At the standard acquisition rate (one housekeeping data set every 30 s) the last 1200 seconds (20 minutes) of housekeeping data are stored in memory.

The average data flow from Amica to ground will be 86 bytes/s, built with 84 bytes/s for pointing data and 87 bytes/min for housekeeping data.

3.9. Serial communication

When the main program is running, the serial communication is managed using IRQ1 and IRQ3 interrupts (during system startup, on the contrary, it is managed asynchronously). During data reception, IRQ3 interrupt means that input serial data is present and ready for CPU, but not yet read. During data transmission, IRQ1 means that output serial data has been read by UART, even if not yet transmitted.

When serial data is received, the program immediately feeds a FIFO to avoid data loss; in fact, each character is overwritten by the subsequent one. When serial data is transmitted, another FIFO, previously fed, is flushed in a single operation.

Amica is connected to two redundant USCM (Universal Slow Control Motion) through RS-232 lines. The maximum speed on the main RS-232 line is 19200 bps, i.e. 2400 characters per second. The second line is capable of a 9600 bps rate.

3.10. Commands Interpreter

Commands built in the receiving serial FIFO are interpreted and sent to a dedicated command buffer. The identifier of each command is compared with a table, and each recognized command is associated with a priority level and an expected execution time. Commands with higher priority (“immediate commands”) are executed at the end of the main loop or, for emergency commands, immediately.

A basic subset of commands has been implemented also in Loader program; these commands are described in section 2.2. The set of commands implemented for attitude determination managing is described in the following subsections.

3.10.1. Pointing Data request command

The request for an updated pointing direction received from USCM is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	11
2	Command ID	word 0	
3	Command ID	word 1	
4	type of pointing data	byte	
5	time from sync ms	long 0	
6	time from sync ms	long 1	
7	time from sync ms	long 2	
8	time from sync ms	long 3	
9	crc	word 0	
10	crc	word 1	

Bytes 5..8 are the relative time in ms of the pointing data request; the fourth bit can be used to specify the desired format of pointing data.

3.10.2. Pointing Data output

The answer to a request for an updated pointing direction received from USCM is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	85
2	message ID	word 0	
3	message ID	word 1	
4	time from sync ms	long 0	
5	time from sync ms	long 1	
6	time from sync ms	long 2	
7	time from sync ms	long 3	
8	abs time s	long 0	seconds elapsed from 1/1/2003
9	abs time s	long 1	(JD-2452640.5)*3600*24
10	abs time s	long 2	
11	abs time s	long 3	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 24/212
 File: AmicaSoftware.doc

12	abs time 4 ms	byte	fraction in 1/256 s
13	ASTC (0 or 1)	byte	0=starboard, 1=port direction
14	camera z RA	word24 0	1/10 of arcsec 0..360
15	camera z RA	word24 1	
16	camera z RA	word24 2	
17	camera z DEC	word24 0	
18	camera z DEC	word24 1	
19	camera z DEC	word24 2	
20	camera x RA	word24 0	
21	camera x RA	word24 1	
22	camera x RA	word24 2	
23	camera x DEC	word24 0	
24	camera x DEC	word24 1	
25	camera x DEC	word24 2	
26	camera y RA	word24 0	
27	camera y RA	word24 1	
28	camera y RA	word24 2	
29	camera y DEC	word24 0	
30	camera y DEC	word24 1	
31	camera y DEC	word24 2	
32	nr of stars in FOV	byte	
33	background	byte 0	measured background/10
34	dark	byte 1	
35	temperature	byte	+/-127 °C
36	star x 0	word 0	in 1/100 of pixel
37	star x 0	word 1	
38	star y 0	word 0	
39	star y 0	word 1	
40	star mgv 0	byte	INT(Log ₂ (intensity)*1000)
41	star x 1	word 0	
42	star x 1	word 1	
43	star y 1	word 0	
44	star y 1	word 1	
45	star mgv 1	byte	
46	star x 2	word 0	
47	star x 2	word 1	
48	star y 2	word 0	
49	star y 2	word 1	
50	star mgv 2	byte	
51	star x 3	word 0	
52	star x 3	word 1	
53	star y 3	word 0	
54	star y 3	word 1	
55	star mgv 3	byte	
56	star x 4	word 0	
57	star x 4	word 1	
58	star y 4	word 0	
59	star y 4	word 1	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 25/212
 File: AmicaSoftware.doc

60	star mgv 4	byte	
61	ms delay last rec	word24 0	
62	ms delay last rec	word24 1	
63	ms delay last rec	word24 2	
64	last ASTC (0 or 1)	byte	
65	camera last rec z RA	word24 0	
66	camera last rec z RA	word24 1	
67	camera last rec z RA	word24 2	
68	camera last rec z DEC	word24 0	
69	camera last rec z DEC	word24 1	
70	camera last rec z DEC	word24 2	
71	camera last rec x RA	word24 0	
72	camera last rec x RA	word24 1	
73	camera last rec x RA	word24 2	
74	camera last rec x DEC	word24 0	
75	camera last rec x DEC	word24 1	
76	camera last rec x DEC	word24 2	
77	camera last rec y RA	word24 0	
78	camera last rec y RA	word24 1	
79	camera last rec y RA	word24 2	
80	camera last rec y DEC	word24 0	
81	camera last rec y DEC	word24 1	
82	camera last rec y DEC	word24 2	
83	crc	word 0	
84	crc	word 1	

Pointing Direction Data Output (long version)

This is the long version of the answer. In bytes 4..7 there is the time elapsed between the last sync and the image to which the pointing data is referred (in milliseconds). The absolute time elapsed from 1/1/2003 is in bytes 8..12.

RA and DEC of the reference axes of the selected camera (ASTC0 or ASTC1), represented in 1/10 of arcseconds, are in bytes 14..31. Each value is three bytes long.

There are also informations on the image content: the number of stars found in the field of view, the background (1/10 of the measured value), the CCD dark noise mean value, the CCD temperature. Coordinates of five reference stars in the image are reported, expressed in 1/100 of pixel; also relative magnitude values are reported.

Finally, informations on the last recognition are reported: delay in milliseconds from the last recognition (bytes 61..63), corresponding camera (byte 64), RA and DEC of last recognition, again expressed in 1/10 of arcseconds.

There are also two alternate short versions for the pointing direction data message, selected by byte 4 in the request command. The first one is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	38
2	message ID	word 0	
3	message ID	word 1	
4	time from sync ms	long 0	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 26/212
 File: AmicaSoftware.doc

5	time from sync ms	long 1	
6	time from sync ms	long 2	
7	time from sync ms	long 3	
8	abs time s	long 0	seconds elapsed from 1/1/2003
9	abs time s	long 1	(JD-2452640.5)*3600*24
10	abs time s	long 2	
11	abs time s	long 3	
12	abs time 4 ms	byte	fraction in 1/256 s
13	ASTC (0 or 1)	byte	0=starboard, 1=port direction
14	camera z RA	word24 0	1/10 of arcsec 0..360
15	camera z RA	word24 1	
16	camera z RA	word24 2	
17	camera z DEC	word24 0	
18	camera z DEC	word24 1	
19	camera z DEC	word24 2	
20	camera x RA	word24 0	
21	camera x RA	word24 1	
22	camera x RA	word24 2	
23	camera x DEC	word24 0	
24	camera x DEC	word24 1	
25	camera x DEC	word24 2	
26	camera y RA	word24 0	
27	camera y RA	word24 1	
28	camera y RA	word24 2	
29	camera y DEC	word24 0	
30	camera y DEC	word24 1	
31	camera y DEC	word24 2	
32	nr of stars in FOV	byte	
33	background	byte 0	measured background/10
34	dark	byte 1	
35	temperature	byte	+/-127 °C
36	crc	word 0	
37	crc	word 1	

Pointing Direction Data Output (short version 1)

The second one is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	45
2	message ID	word 0	
3	message ID	word 1	
4	time from sync ms	long 0	
5	time from sync ms	long 1	
6	time from sync ms	long 2	
7	time from sync ms	long 3	
8	abs time s	long 0	seconds elapsed from 1/1/2003
9	abs time s	long 1	(JD-2452640.5)*3600*24

10	abs time s	long 2	
11	abs time s	long 3	
12	abs time 4 ms	byte	fraction in 1/256 s
13	ASTC (0 or 1)	byte	0=starboard, 1=port direction
14	nr of stars in FOV	byte	
15	background	byte 0	measured background/10
16	dark	byte 1	
17	temperature	byte	+/-127 °C
18	star x 0	word 0	in 1/100 of pixel
19	star x 0	word 1	
20	star y 0	word 0	
21	star y 0	word 1	
22	star mgv 0	byte	INT(Log ₂ (intensity)*1000)
23	star x 1	word 0	
24	star x 1	word 1	
25	star y 1	word 0	
26	star y 1	word 1	
27	star mgv 1	byte	
28	star x 2	word 0	
29	star x 2	word 1	
30	star y 2	word 0	
31	star y 2	word 1	
32	star mgv 2	byte	
33	star x 3	word 0	
34	star x 3	word 1	
35	star y 3	word 0	
36	star y 3	word 1	
37	star mgv 3	byte	
38	star x 4	word 0	
39	star x 4	word 1	
40	star y 4	word 0	
41	star y 4	word 1	
42	star mgv 4	byte	
43	crc	word 0	
44	crc	word 1	

Pointing Direction Data Output (short version 2)

Finally, a very simple version of output message, where only a status byte and the time elapsed from last sync are reported, is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	11
2	message ID	word 0	
3	message ID	word 1	
4	time from sync ms	long 0	
5	time from sync ms	long 1	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 28/212
 File: AmicaSoftware.doc

6	time from sync ms	long 2	
7	time from sync ms	long 3	
8	status	byte	
9	crc	word 0	
10	crc	word 1	

Alive Data Output

3.10.3. Attitude Quaternion command

The message in which a new ISS attitude information is sent to Amica is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr_bytes	byte	27
2	Command ID	word 0	
3	Command ID	word 1	
4	q0	float 0	
5	q0	float 1	
6	q0	float 2	
7	q0	float 3	
8	q1	float 0	
9	q1	float 1	
10	q1	float 2	
11	q1	float 3	
12	q2	float 0	
13	q2	float 1	
14	q2	float 2	
15	q2	float 3	
16	q3	float 0	
17	q3	float 1	
18	q3	float 2	
19	q3	float 3	
20	attitude time s	long 0	seconds elapsed from 1/1/2003
21	attitude time s	long 1	(JD-2452640.5)*3600*24
22	attitude time s	long 2	
23	attitude time s	long 3	
24	attitude time 4 ms	byte	fraction in 1/256 s
25	crc	word 0	
26	crc	word 1	

This new attitude information, in which the quaternion is represented using four float values, will be the new seed for precise attitude calculation based on star field recognition. Four bytes are used to represent the starting time of the new attitude, as the number of seconds elapsed from 1/1/2003. An additional byte represents the number of milliseconds, compressed in the 0..255 range.

3.10.4. Attitude Roll-Pitch-Yaw command

An alternate way to send ISS attitude to Amica using Roll, Pitch and Yaw is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	24
2	Command ID	word 0	
3	Command ID	word 1	
4	sequence	byte	1=YRP, 2=RPY, ...
5	roll	float 0	
6	roll	float 1	
7	roll	float 2	
8	roll	float 3	
9	pitch	float 0	
10	pitch	float 1	
11	pitch	float 2	
12	pitch	float 3	
13	yaw	float 0	
14	yaw	float 1	
15	yaw	float 2	
16	yaw	float 3	
17	attitude time s	long 0	seconds elapsed from 1/1/2003
18	attitude time s	long 1	(JD-2452640.5)*3600*24
19	attitude time s	long 2	
20	attitude time s	long 3	
21	attitude time 4 ms	byte	fraction in 1/256 s
22	crc	word 0	
23	crc	word 1	

3.10.5. Orbit TLE command

The Two Lines Elements representation of ISS orbit is sent to Amica using this command:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	144
2	Command ID	word 0	
3	Command ID	word 1	
4	TLE first line byte 1	byte	
...	
72	TLE first line byte 69	byte	
73	TLE second line byte 1	byte	
...	
141	TLE second line byte 69	byte	
142	crc	word 0	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 30/212
 File: AmicaSoftware.doc

143	crc	word 1	
-----	-----	--------	--

3.10.6. Time command

With this command a new absolute time value is sent to Amica. This time is corresponding to the last Sync signal sent on LVDS line.

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	11
2	Command ID	word 0	
3	Command ID	word 1	
4	attitude time s	long 0	seconds elapsed from 1/1/2003
5	attitude time s	long 1	(JD-2452640.5)*3600*24
6	attitude time s	long 2	
7	attitude time s	long 3	
8	attitude time 4 ms	byte	fraction in 1/256 s
9	crc	word 0	
10	crc	word 1	

3.10.7. Housekeeping data

The answer to a request for Amica housekeeping data, received from USCM once per minute, is structured as follows:

byte	description	type	notes
0	sync	byte	0xE5
1	message nr bytes	byte	88
2	message ID	word 0	
3	message ID	word 1	
4	abs time s	long 0	seconds elapsed from 1/1/2003
5	abs time s	long 1	(JD-2452640.5)*3600*24
6	abs time s	long 2	
7	abs time s	long 3	
8	abs time 4 ms	byte	fraction in 1/256 s
9	ms start interval	word24 0	16777.216 s
10	ms start interval	word24 1	
11	ms start interval	word24 2	
12	temperature1 astc0	word 0	
13	temperature1 astc0	word 1	
14	temperature2 astc0	word 0	
15	temperature2 astc0	word 1	
16	current1 astc0	word 0	
17	current1 astc0	word 1	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 31/212
 File: AmicaSoftware.doc

18	current2 astc0	word 0	
19	current2 astc0	word 1	
20	voltage1 astc0	word 0	
21	voltage1 astc0	word 1	
22	voltage2 astc0	word 0	
23	voltage2 astc0	word 1	
24	status1 astc0	word 0	
25	status1 astc0	word 1	
26	status2 astc0	word 0	
27	status2 astc0	word 1	
28	time1 astc0	word 0	
29	time1 astc0	word 1	
30	phot1 astc0	word 0	
31	phot1 astc0	word 1	
32	phot2 astc0	word 0	
33	phot2 astc0	word 1	
34	phot3 astc0	word 0	
35	phot3 astc0	word 1	
36	phot4 astc0	word 0	
37	phot4 astc0	word 1	
38	temperature1 astc1	word 0	
39	temperature1 astc1	word 1	
40	temperature2 astc1	word 0	
41	temperature2 astc1	word 1	
42	current1 astc1	word 0	
43	current1 astc1	word 1	
44	current2 astc1	word 0	
45	current2 astc1	word 1	
46	voltage1 astc1	word 0	
47	voltage1 astc1	word 1	
48	voltage2 astc1	word 0	
49	voltage2 astc1	word 1	
50	status1 astc1	word 0	
51	status1 astc1	word 1	
52	status2 astc1	word 0	
53	status2 astc1	word 1	
54	time1 astc1	word 0	
55	time1 astc1	word 1	
56	phot1 astc1	word 0	
57	phot1 astc1	word 1	
58	phot2 astc1	word 0	
59	phot2 astc1	word 1	
60	phot3 astc1	word 0	
61	phot3 astc1	word 1	
62	phot4 astc1	word 0	
63	phot4 astc1	word 1	
64	status1_e aste	word 0	
65	status1_e aste	word 1	



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
 ISSUE: 1
 DATE: Jun 07
 PAGE: 32/212
 File: AmicaSoftware.doc

66	status2_e_aste	word 0	
67	status2_e_aste	word 1	
68	status3_e_aste	word 0	
69	status3_e_aste	word 1	
70	status4_e_aste	word 0	
71	status4_e_aste	word 1	
72	temperature1_aste	word 0	
73	temperature1_aste	word 1	
74	temperature2_aste	word 0	
75	temperature2_aste	word 1	
76	voltage_aste	word 0	
77	voltage_aste	word 1	
78	status1_sw_aste	word 0	
79	status1_sw_aste	word 1	
80	status2_sw_aste	word 0	
81	status2_sw_aste	word 1	
82	status3_sw_aste	word 0	
83	status3_sw_aste	word 1	
84	status4_sw_aste	word 0	
85	status4_sw_aste	word 1	
86	crc	word 0	
87	crc	word 1	

Housekeeping Data Output

4. SIMULATIONS

A fully parametrized software generator of artificial images has been developed. The 512x512 pixels images generated have been used to test the algorithms for star searching and star fields recognition.

The generator is able to add noise, a nonzero mean level due to sky background, and also image artifacts, e.g rows or bad pixels. These options have been useful to deeply test the algorithms in conditions similar to the real ones.

The stars generator can operate with various point-spread-function parameters, using several different psf models (Gaussian, asymmetrical Gaussian, Moffat) and various FWHM values (full width at half maximum), simulating defocusing effects.

The centroiding algorithm has been tested over different conditions, with star images with different FWHM value; some samples of such images are shown in *Figure.5*.

Also pointing determination algorithms can be tested on images generated by the same simulator; for this purpose, the generator takes star field coordinates from the full version of the catalogue used on-board; in this way, the effect of the presence in the artificial image also of fainter stars, not present in the on-board catalogue, can be taken into account, simulating real operation of the system. The stars taken from the catalogue are re-projected on the plane tangent to the celestial sphere at its intersection with the pointing direction; star coordinates (RA, DEC) from the catalog are converted to pixel coordinates on the CCD plane.

Figure.1 - Startup procedure flowchart

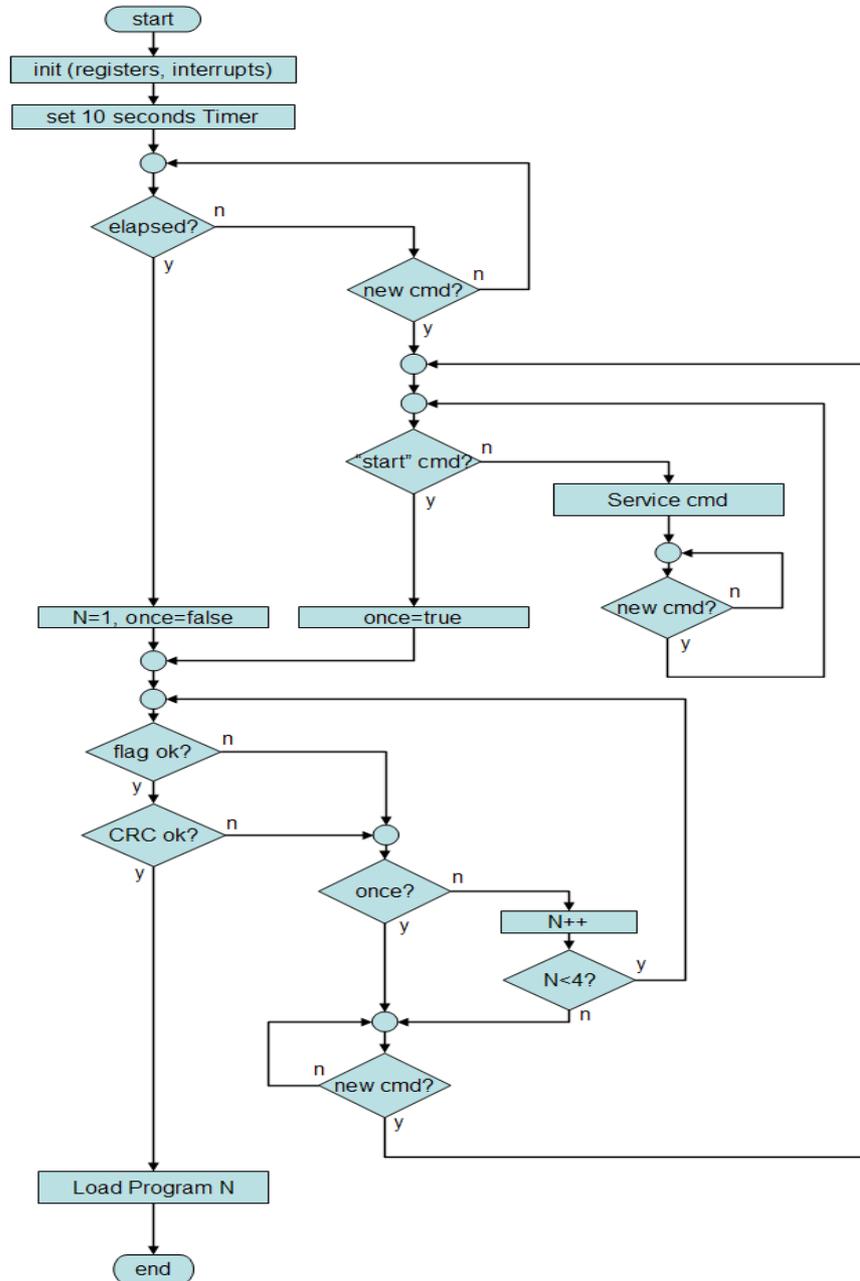


Figure.2 – Tracker Main process and Auxiliary processes

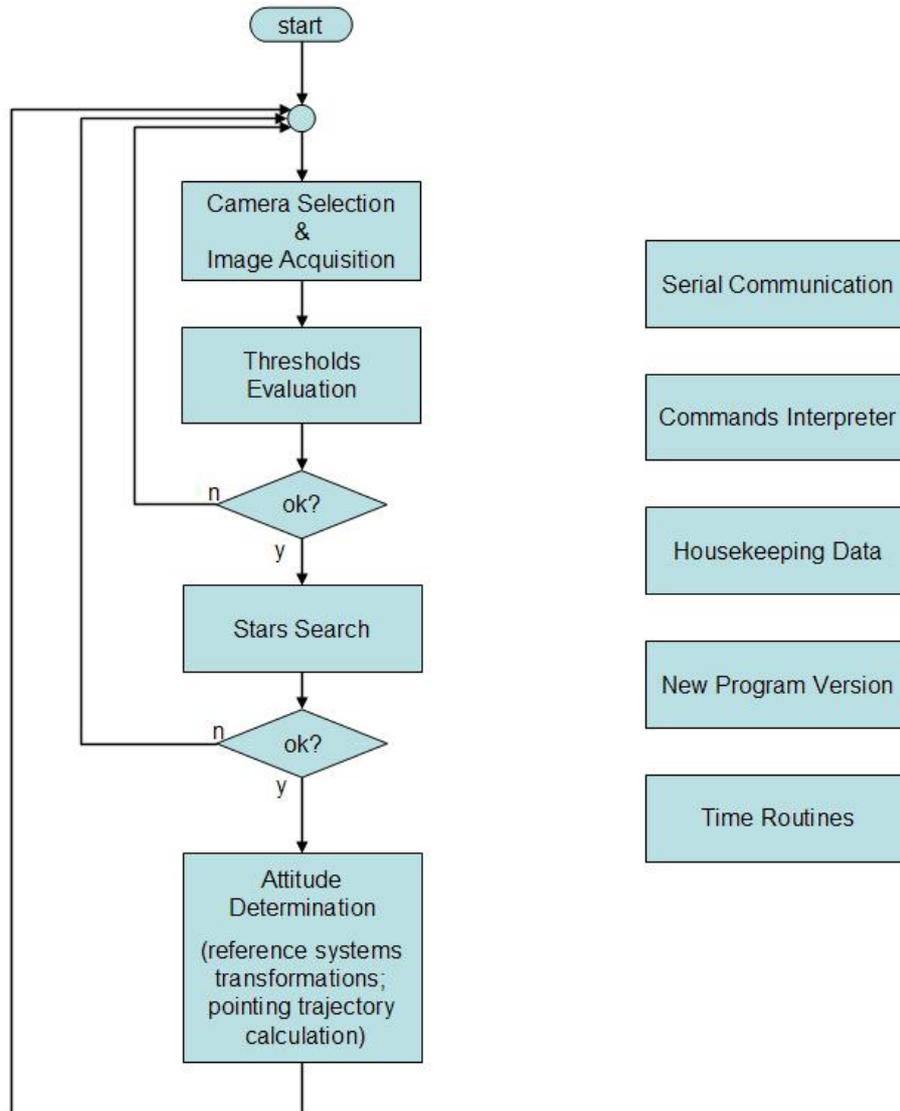


Figure.3 - Flash program memory and Ram program memory usage

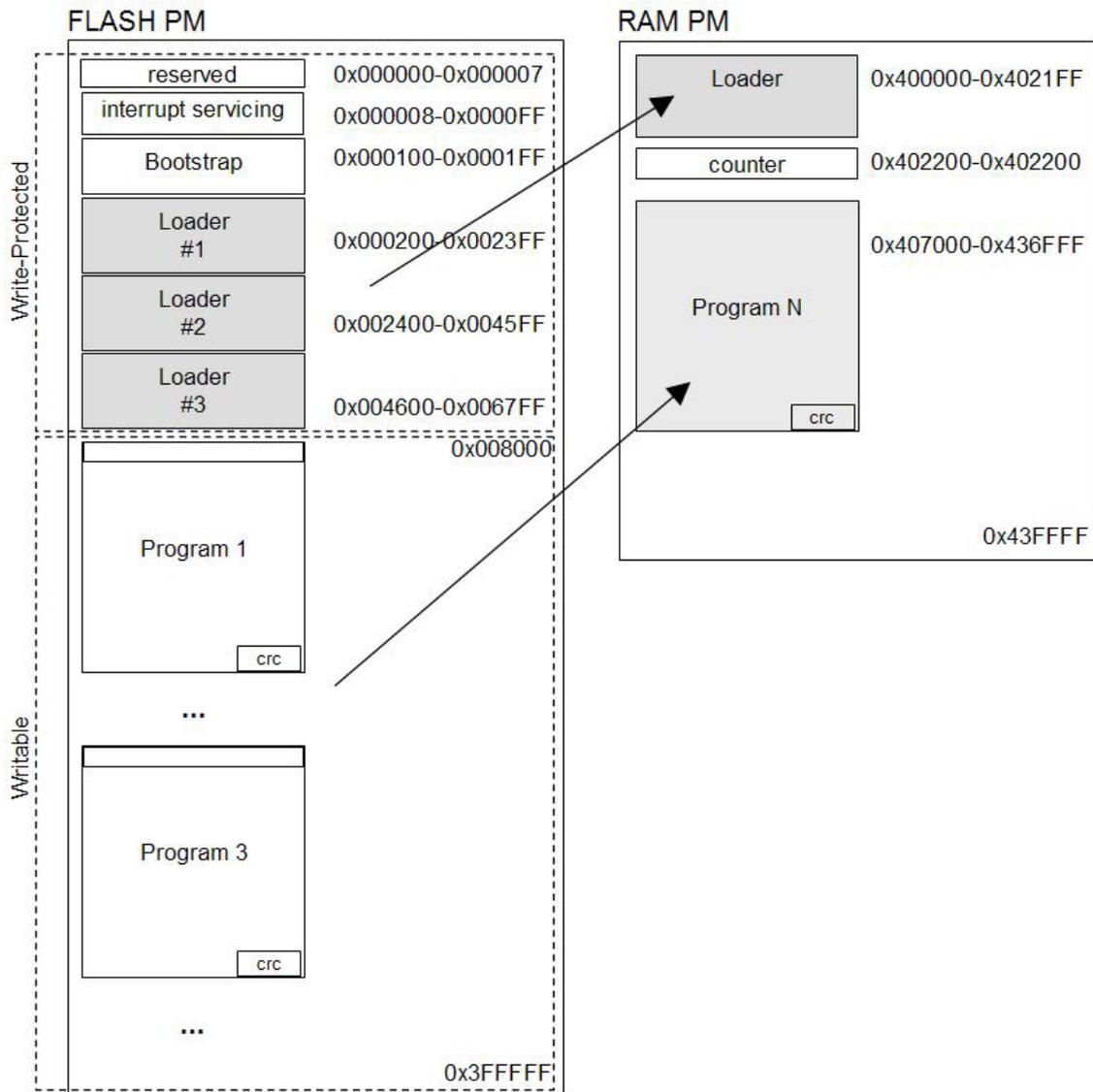


Figure.4 – Search and Track main loop

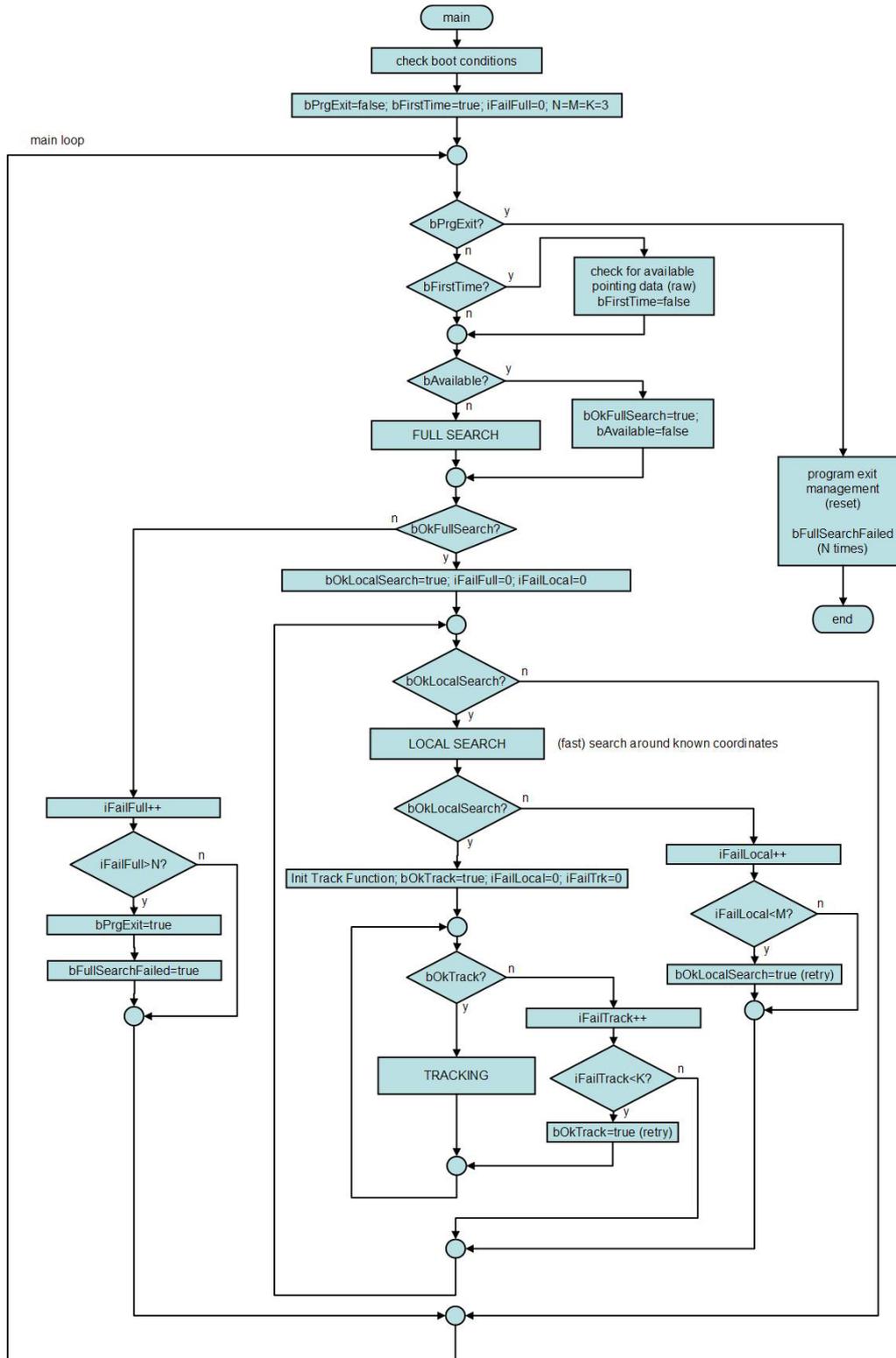
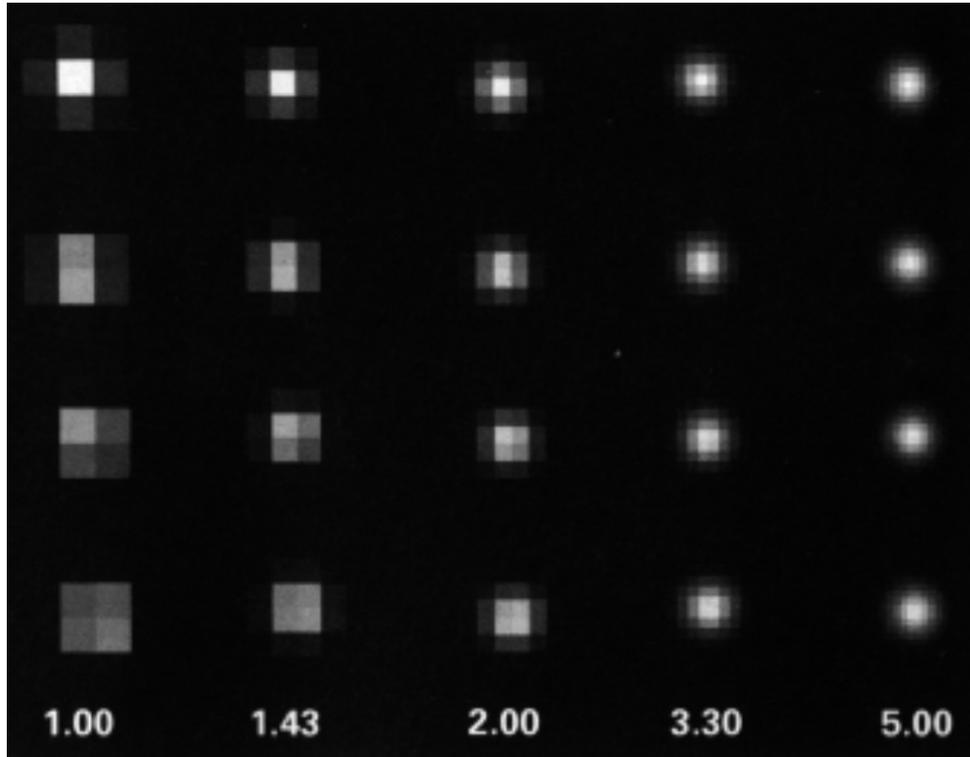


Figure.5 - artificial star images, with various FWHM values





AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 38/212
File: AmicaSoftware.doc

5. SOURCE FILES

5.1. Bootstrap

5.1.1. bootstrp.ach

```
.system bootstrap
.processor=ADSP21020;

!***** Program memory *****/
!vector table in flash
.SEGMENT /ROM /BEGIN=0x000000 /END=0x0000ff /PM seg_rt1;

!bootstrap program code in flash
.SEGMENT /ROM /BEGIN=0x000100 /END=0x0001ff /PM pm_bsc0;
!loader program code in flash - three copies
.SEGMENT /ROM /BEGIN=0x000200 /END=0x0067ff /PM pm_ldco;

!main program code in flash - three copies
.SEGMENT /RAM /BEGIN=0x008000 /END=0x00afff /PM pm_mpc0;
!star catalogue in flash - two copies
.SEGMENT /RAM /BEGIN=0x0b0000 /END=0x022fff /PM pm_cats;

!loader vector table in ram
.SEGMENT /RAM /BEGIN=0x400000 /END=0x4000ff /PM seg_rth;
!loader program code in ram
.SEGMENT /RAM /BEGIN=0x400100 /END=0x4020ff /PM seg_pmco;
.SEGMENT /RAM /BEGIN=0x402100 /END=0x4021ff /PM seg_init;
.SEGMENT /RAM /BEGIN=0x402200 /END=0x402200 /PM pm_ldcnt;
.SEGMENT /RAM /BEGIN=0x402201 /END=0x402201 /PM seg_pmda;

!***** Data memory *****/
!data ram a 40 bit
.SEGMENT /RAM /BEGIN=0x00000000 /END=0x0003ffff /DM dm_bank0;
!data ram a 16 bit
.SEGMENT /RAM /BEGIN=0x20000000 /END=0x2003ffff /DM dm_bank1;
!ram immagini
.SEGMENT /RAM /BEGIN=0x40000000 /END=0x4003ffff /DM dm_bank2;
!porte i/o
.SEGMENT /RAM /BEGIN=0x80000000 /END=0x800000ff /DM dm_bank3;

.ENDSYS;
```

5.1.2. bootstrp.asm

////////////////////////////////////



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 39/212
File: AmicaSoftware.doc

```
//  
// AMICA FOR AMS  
//  
// File: bootstrp.asm  
//  
// Version: 1.0  
// Modified: 30/05/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: Bootstrap code  
// Loads modal version of LOADER program to PM RAM starting at 0x400000  
// (execution will start at 0x400100)  
//  
////////////////////////////////////  
  
#include "def21020.h"  
#include "defboot.h"  
  
.SEGMENT /PM pm_bsco;  
begin: pmwait = b#00000000101101; /* PM wait states: 0ws b1, 3ws b0 */  
dmwait = b#000000101001010000100001;  
/* DM wait states: 1ws b3, 1ws b2, 0ws b1, 0ws b0 */  
bit clr MODE1 IRPTEN; /* disable interrupts */  
/* intialization */  
i8= L1START; /* pointer to Loader program #1 */  
m8=1; /* M8 increments by 1 */  
l8=0; /* no circular buffer */  
i9= L2START; /* pointer to Loader program #2 */  
m9=1; /* M9 increments by 1 */  
l9=0; /* no circular buffer */  
i10=L3START; /* pointer to Loader program #3 */  
m10=1; /* M10 increments by 1 */  
l10=0; /* no circular buffer */  
i11=LMSTART; /* load majority program at LMSTART */  
m11=1; /* M11 increments by 1 */  
l11=0; /* no circular buffer */  
r8=0x00; /* Loader majority exceptions counter */  
pm(COUNT1)=r8; /* reset majority exceptions counter */  
lcntr=LDRLEN, do maj_test until lce;  
px=pm(i8,m8); /* load instruction from Loader #1 */  
r0=px1;  
r1=px2;  
px=pm(i9,m9); /* load instruction from Loader #2 */  
r2=px1;  
r3=px2;  
px=pm(i10,m10); /* load instruction from Loader #3 */  
r4=px1;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 40/212
File: AmicaSoftware.doc

```
r5=px2;
/* maj = ((p1 xor p2)and(p3)) or ((not(p1 xor p2))and(p1)) */
r6=r0 xor r2;          /* 16-bit; compare #1 and #2 */
if eq jump px1ok;
nop;
nop;
r8=r8+1;              /* increment error counter */
r7=r6 and r4;
r6=not r6;
r6=r6 and r0;
r0=r7 or r6;          /* reuse register r0 */
px1ok: px1=r0;         /* majority px1 value */
r6=r1 xor r3;         /* 32-bit; compare #1 and #2 */
if eq jump px2ok;
nop;
nop;
r8=r8+1;              /* increment error counter */
r7=r6 and r5;
r6=not r6;
r6=r6 and r1;
r1=r7 or r6;         /* reuse register r1 */
px2ok: px2=r1;        /* majority px2 value */
pm(i11,m11)=px;      /* write current instruction */
nop;
pm(COUNT1)=r8;       /* write error counter value */
nop;
maj_test: nop;
nop;
nop;
jump LMBEGIN;        /* begin Loader program execution */
.ENDSEG;
```

```
/* The error correction is made bit-to-bit, based on a          */
/* majority check over the three copies of Loader program.     */
/* There is a unique counter for the total number of the      */
/* majority errors found. The counter value is the number     */
/* of PX1 or PX2 registers found with a wrong value (1 or    */
/* more bits, typically 1) that has been corrected by the    */
/* majority algorithm.                                         */
/* An estimate of the number of errors found in each of       */
/* three copies of the Loader program is possible but         */
/* onerous compared to the real benefits (in fact the        */
/* computational charge would be doubled).                    */
```

5.1.3. def21020.h



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 41/212
File: AmicaSoftware.doc

```
*
* def21020.h
* STATUS REGISTER BIT DEFINITIONS FOR ADSP-21020.
*
* (c) Copyright 2001-2004 Analog Devices, Inc. All rights reserved.
* $Revision: 1.1 $
*****/
/*
This include file contains a list of "defines" to enable the programmer to
use symbolic names for all of the system register bits for the ADSP-21020.
*/
#ifndef __DEF21020_H_
#define __DEF21020_H_

/* MODE1 register */
#define BR0 0x00000002 /* Bit 1: Bit-reverse for I0 (uses DMS0- only) */
#define SRCU 0x00000004 /* Bit 2: Alt. register select for comp. units */
#define SRD1H 0x00000008 /* Bit 3: DAG1 alt. register select (7-4) */
#define SRD1L 0x00000010 /* Bit 4: DAG1 alt. register select (3-0) */
#define SRD2H 0x00000020 /* Bit 5: DAG2 alt. register select (15-12) */
#define SRD2L 0x00000040 /* Bit 6: DAG2 alt. register select (11-8) */
#define SRRFH 0x00000080 /* Bit 7: Register file alt. select for R(15-8) */
#define SRRFL 0x00000400 /* Bit 10: Register file alt. select for R(7-0) */
#define NESTM 0x00000800 /* Bit 11: Interrupt nesting enable */
#define IRPTEN 0x00001000 /* Bit 12: Global interrupt enable */
#define ALUSAT 0x00002000 /* Bit 13: Enable ALU fixed-pt. saturation */
#define TRUNC 0x00008000 /* Bit 15: 1=flt-pt. truncation 0=Rnd to nearest */
#define RND32 0x00010000 /* Bit 16: 1=32-bit fltg-pt.rounding 0=40-bit rnd */

/* MODE2 register */
#define IRQ0E 0x00000001 /* Bit 0: IRQ0- 1=edge sens. 0=level sens. */
#define IRQ1E 0x00000002 /* Bit 1: IRQ1- 1=edge sens. 0=level sens. */
#define IRQ2E 0x00000004 /* Bit 2: IRQ2- 1=edge sens. 0=level sens. */
#define IRQ3E 0x00000008 /* Bit 3: IRQ3- 1=edge sens. 0=level sens. */
#define CADIS 0x00000010 /* Bit 4: Cache disable */
#define TIMEN 0x00000020 /* Bit 5: Timer enable */
#define FLG0O 0x00008000 /* Bit 15: FLAG0 1=output 0=input */
#define FLG1O 0x00010000 /* Bit 16: FLAG1 1=output 0=input */
#define FLG2O 0x00020000 /* Bit 17: FLAG2 1=output 0=input */
#define FLG3O 0x00040000 /* Bit 18: FLAG3 1=output 0=input */
#define CAFRZ 0x00080000 /* Bit 19: Cache freeze */

/* ASTAT register */
#define AZ 0x00000001 /* Bit 0: ALU result zero or fltg-pt. underflow */
#define AV 0x00000002 /* Bit 1: ALU overflow */
#define AN 0x00000004 /* Bit 2: ALU result negative */
#define AC 0x00000008 /* Bit 3: ALU fixed-pt. carry */
#define AS 0x00000010 /* Bit 4: ALU X input sign (ABS and MANT ops) */
```

```

#define AI    0x00000020 /* Bit 5: ALU fltg-pt. invalid operation */
#define MN    0x00000040 /* Bit 6: Multiplier result negative */
#define MV    0x00000080 /* Bit 7: Multiplier overflow */
#define MU    0x00000100 /* Bit 8: Multiplier fltg-pt. underflow */
#define MI    0x00000200 /* Bit 9: Multiplier fltg-pt. invalid operation */
#define AF    0x00000400 /* Bit 10: ALU fltg-pt. operation */
#define SV    0x00000800 /* Bit 11: Shifter overflow */
#define SZ    0x00001000 /* Bit 12: Shifter result zero */
#define SS    0x00002000 /* Bit 13: Shifter input sign */
#define BTF   0x00040000 /* Bit 18: Bit test flag for system registers */
#define FLG0  0x00080000 /* Bit 19: FLAG0 value */
#define FLG1  0x00100000 /* Bit 20: FLAG1 value */
#define FLG2  0x00200000 /* Bit 21: FLAG2 value */
#define FLG3  0x00400000 /* Bit 22: FLAG3 value */
#define CACC0 0x01000000 /* Bit 24: Compare Accumulation Bit 0 */
#define CACC1 0x02000000 /* Bit 25: Compare Accumulation Bit 1 */
#define CACC2 0x04000000 /* Bit 26: Compare Accumulation Bit 2 */
#define CACC3 0x08000000 /* Bit 27: Compare Accumulation Bit 3 */
#define CACC4 0x10000000 /* Bit 28: Compare Accumulation Bit 4 */
#define CACC5 0x20000000 /* Bit 29: Compare Accumulation Bit 5 */
#define CACC6 0x40000000 /* Bit 30: Compare Accumulation Bit 6 */
#define CACC7 0x80000000 /* Bit 31: Compare Accumulation Bit 7 */

/* STKY register */
#define AUS   0x00000001 /* Bit 0: ALU fltg-pt. underflow */
#define AVS   0x00000002 /* Bit 1: ALU fltg-pt. overflow */
#define AOS   0x00000004 /* Bit 2: ALU fixed-pt. overflow */
#define AIS   0x00000020 /* Bit 5: ALU fltg-pt. invalid operation */
#define MOS   0x00000040 /* Bit 6: Multiplier fixed-pt. overflow */
#define MVS   0x00000080 /* Bit 7: Multiplier fltg-pt. overflow */
#define MUS   0x00000100 /* Bit 8: Multiplier fltg-pt. underflow */
#define MIS   0x00000200 /* Bit 9: Multiplier fltg-pt. invalid operation */
#define CB7S  0x00020000 /* Bit 17: DAG1 circular buffer 7 overflow */
#define CB15S 0x00040000 /* Bit 18: DAG2 circular buffer 15 overflow */
#define PCFL  0x00200000 /* Bit 21: PC stack full */
#define PCEM  0x00400000 /* Bit 22: PC stack empty */
#define SSOV  0x00800000 /* Bit 23: Status stack overflow (MODE1 and ASTAT) */
#define SSEM  0x01000000 /* Bit 24: Status stack empty */
#define LSOV  0x02000000 /* Bit 25: Loop stack overflow */
#define LSEM  0x04000000 /* Bit 26: Loop stack empty */

/* IRPTL and IMASK and IMASKP registers */
#define RSTI  0x00000002 /* Bit 1: Address: 08: Reset */
#define SOVFI 0x00000008 /* Bit 3: Address: 18: Stack overflow */
#define TMZHI 0x00000010 /* Bit 4: Address: 20: Timer = 0 (high priority) */
#define IRQ3I 0x00000020 /* Bit 5: Address: 28: IRQ3- asserted */
#define IRQ2I 0x00000040 /* Bit 6: Address: 30: IRQ2- asserted */
#define IRQ1I 0x00000080 /* Bit 7: Address: 38: IRQ1- asserted */
#define IRQ0I 0x00000100 /* Bit 8: Address: 40: IRQ0- asserted

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 43/212
File: AmicaSoftware.doc

```
#define CB7I 0x0000800 /* Bit 11: Address: 58: Circ. buffer 7 overflow */
#define CB15I 0x00001000 /* Bit 12: Address: 60: Circ. buffer 15 overflow */
#define TMZLI 0x00004000 /* Bit 14: Address: 70: Timer = 0 (low priority) */
#define FIXI 0x00008000 /* Bit 15: Address: 78: Fixed-pt. overflow */
#define FLTOI 0x00010000 /* Bit 16: Address: 80: fltg-pt. overflow */
#define FLTUI 0x00020000 /* Bit 17: Address: 88: fltg-pt. underflow */
#define FLTII 0x00040000 /* Bit 18: Address: 90: fltg-pt. invalid */
#define SFT0I 0x01000000 /* Bit 24: Address: C0: user software int 0 */
#define SFT1I 0x02000000 /* Bit 25: Address: C8: user software int 1 */
#define SFT2I 0x04000000 /* Bit 26: Address: D0: user software int 2 */
#define SFT3I 0x08000000 /* Bit 27: Address: D8: user software int 3 */
#define SFT4I 0x10000000 /* Bit 28: Address: E0: user software int 4 */
#define SFT5I 0x20000000 /* Bit 29: Address: E8: user software int 5 */
#define SFT6I 0x40000000 /* Bit 30: Address: F0: user software int 6 */
#define SFT7I 0x80000000 /* Bit 31: Address: F8: user software int 7 */

#endif // __DEF21020_H_

/* end of file */
```

5.1.4. defboot.h

```
////////////////////////////////////
//
// AMICA FOR AMS
//
// File: defboot.h
//
// Version: 1.0
// Modified: 30/05/07
// Author: F.Roncella
// Hardware: ADSP-21020
// Project: StarTracker
//
// Notes: Bootstrap code definitions
//
////////////////////////////////////

#ifndef __DEFBOOT_H_
#define __DEFBOOT_H_

/* Flash Memory */
#define BSSTART 0x000100 /* Bootstrap code starting address */
#define BSEND 0x0001FF /* Bootstrap code ending address */
#define L1START 0x000200 /* Loader code #1 starting address */
#define L1END 0x0023FF /* Loader code #1 ending address */
#define L2START 0x002400 /* Loader code #2 starting address */
#define L2END 0x0045FF /* Loader code #2 ending address */
#define L3START 0x004600 /* Loader code #3 starting address */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 44/212
File: AmicaSoftware.doc

```
#define L3END      0x0067FF      /* Loader code #3 ending address */  
#define LDRLLEN   0x002200      /* Loader code length */  
  
/* RAM Memory */  
#define LMSTART   0x400000      /* Loader code #4 (majority) initial address */  
#define LMEND     0x4021FF      /* Loader code #4 (majority) ending address */  
#define LMBEGIN   0x400100      /* Loader code #4 start execution address */  
#define COUNT1    0x402200      /* Loader majority exceptions counter */  
  
#endif // DEFBOOT H
```

5.2. Loader



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 45/212
File: AmicaSoftware.doc

5.2.1. Loader.ach

```
.system loader
.processor=ADSP21020;

!***** Program memory *****/
!vector table in flash
!.SEGMENT /ROM /BEGIN=0x000000 /END=0x0000FF /PM seg_rth;
.SEGMENT /ROM /BEGIN=0x000000 /END=0x0000FF /PM seg_rt1;
!.SEGMENT /ROM /BEGIN=0x000100 /END=0x0001FF /PM seg_dum1;

!bootstrap program code in flash
.SEGMENT /ROM /BEGIN=0x000100 /END=0x0001FF /PM pm_bsco;
!loader program code in flash - three copies
.SEGMENT /ROM /BEGIN=0x000200 /END=0x0067FF /PM pm_ldco;

!flash utility writes here:

!main program code in flash - three copies
.SEGMENT /RAM /BEGIN=0x008000 /END=0x0AFFFF /PM pm_mpc;
!star catalogue in flash - two copies
.SEGMENT /RAM /BEGIN=0x0B0000 /END=0x22FFFF /PM pm_cats;
!spare
.SEGMENT /RAM /BEGIN=0x230000 /END=0x3FFFFFF /PM pm_free;

!loader vector table in ram
!.SEGMENT /ROM /BEGIN=0x400000 /END=0x4000FF /PM seg_rth;
.SEGMENT /RAM /BEGIN=0x400000 /END=0x4000FF /PM seg_rth;
!loader program code in ram
.SEGMENT /RAM /BEGIN=0x400100 /END=0x4020FF /PM seg_pmco;
!.SEGMENT /ROM /BEGIN=0x402100 /END=0x4021FF /PM seg_init;
.SEGMENT /RAM /BEGIN=0x402100 /END=0x4021FF /PM seg_init;
.SEGMENT /RAM /BEGIN=0x402200 /END=0x402200 /PM pm_ldcnt;
.SEGMENT /RAM /BEGIN=0x402201 /END=0x402201 /PM seg_pmda;

!main program vector table in ram
.SEGMENT /ROM /BEGIN=0x407000 /END=0x4070FF /PM sg1_rth;
!main program program code in ram
.SEGMENT /RAM /BEGIN=0x407100 /END=0x436EFF /PM pm_mpram;
.SEGMENT /ROM /BEGIN=0x436F00 /END=0x436FFF /PM sg1_init;
.SEGMENT /RAM /BEGIN=0x437000 /END=0x437000 /PM sg1_pmda;

!***** Data memory *****/
!data ram a 40 bit
!.SEGMENT /RAM /BEGIN=0x00000000 /END=0x0003FFFF /DM dm_bank0;

.SEGMENT /RAM /BEGIN=0x00000000 /END=0x000039FF /DM seg_dmda;
.SEGMENT /RAM /BEGIN=0x00003A00 /END=0x000063FF /CSTACK /DM seg_stak;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 46/212
File: AmicaSoftware.doc

```
.SEGMENT /RAM /BEGIN=0x00006400 /END=0x00008FFF /CHEAP /DM seg_heap;  
  
.SEGMENT /RAM /BEGIN=0x0000F000 /END=0x000129FF /DM sg1_dmda;  
.SEGMENT /RAM /BEGIN=0x00012A00 /END=0x000153FF /CSTACK /DM sg1_stak;  
.SEGMENT /RAM /BEGIN=0x00015400 /END=0x00017FFF /CHEAP /DM sg1_heap;  
.SEGMENT /RAM /BEGIN=0x00018000 /END=0x0001FFFF /DM seg_flb;  
  
!data ram a 16 bit  
.SEGMENT /RAM /BEGIN=0x20000000 /END=0x2003FFFF /DM dm_bank1;  
  
!ram immagini  
.SEGMENT /RAM /BEGIN=0x40000000 /END=0x4003FFFF /DM dm_bank2;  
  
!porte i/o  
.SEGMENT /RAM /BEGIN=0x80000000 /END=0x800000FF /DM dm_bank3;  
  
.ENDSYS;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 47/212
File: AmicaSoftware.doc

5.2.2. copy.asm

```
//////////////////////////////////////
//
// AMICA FOR AMS
//
// File:   copy.asm
//
// Version: 1.0
// Modified: 28/05/07
// Author: F.Roncella
// Hardware: ADSP-21020
// Project: StarTracker
//
// Notes:  Main program copy utility
//         Loads program to PM RAM starting at 0x407000
//
//////////////////////////////////////

/*#include <def21020.h>*/
#include <asm_sprt.h>

/* Flash Memory */
#define P1START    0x010000    /* Main program code #1 starting address */
#define P2START    0x048000    /* Main program code #2 starting address */
#define P3START    0x080000    /* Main program code #3 starting address */
//#define LEN    0x00FAAA    /* Main program length 2F000 = FAAA * 3 */
#define LEN    0x00C000    /* Main program length 30000 = C000 * 4 */

/*
   copy 0xC000 words from P1START to MPSTART,
   copy 0xC000 words from P1START+0xC000 to MPSTART+0x0xC000,
   copy 0xC000 words from P1START+0x18000 to MPSTART+0x18000.      */

/* RAM Memory */
#define MPSTART    0x407000    /* Main program code RAM starting address */

.SEGMENT /DM seg_dmda;

.var _asm_var=0;          /* asm_var is defined here */
.global _asm_var;        /* needed so C can see it */

.ENDSEG;

.SEGMENT /PM seg_pmco;

.global _asm_copy;       /* _asm_copy is defined here */
.extern _ivers;          /* _ivers is defined in C file */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 48/212
File: AmicaSoftware.doc

```
_asm_copy:
    leaf_entry;          /* entry macro 4-13 */

    /* r8=dm(_ivers); */ /* access the global C variable */
    /* dm(_asm_var)=r8; */ /* set asm_var to ivers */

    i8=P1START;         /* pointer to Main program #1 */
    m8=1;               /* M8 increments by 1 */
    l8=0;               /* no circular buffer */
    i11=MPSTART;        /* load majority program at MPSTART */
    m11=1;              /* M11 increments by 1 */
    l11=0;              /* no circular buffer */
    lcntr=LEN, do mpcpy1 until lce;
    px=pm(i8,m8);       /* load instruction from Main program #1 */
    pm(i11,m11)=px;    /* write current instruction */
    nop;
    nop;
mpcpy1: nop;
    nop;
    nop;
    lcntr=LEN, do mpcpy2 until lce;
    px=pm(i8,m8);       /* load instruction from Main program #1 */
    pm(i11,m11)=px;    /* write current instruction */
    nop;
    nop;
mpcpy2: nop;
    nop;
    nop;
    lcntr=LEN, do mpcpy3 until lce;
    px=pm(i8,m8);       /* load instruction from Main program #1 */
    pm(i11,m11)=px;    /* write current instruction */
    nop;
    nop;
mpcpy3: nop;
    nop;
    nop;
    leaf_exit;         /* exit macro */
.ENDSEG;
```

5.2.3. copy1.asm

```

////////////////////////////////////
//
// AMICA FOR AMS
//
// File:   copy1.asm
//
// Version: 1.0
// Modified: 28/05/07
// Author: F.Roncella
// Hardware: ADSP-21020
// Project: StarTracker
//
// Notes:  Main program copy utility
//         Loads program to PM RAM starting at 0x407000
//
////////////////////////////////////

/*#include <def21020.h>*/
#include <asm_sprt.h>

/* Flash Memory */
#define P1START    0x010000    /* Main program code #1 starting address */
#define P2START    0x048000    /* Main program code #2 starting address */
#define P3START    0x080000    /* Main program code #3 starting address */
#define LEN        0x00C000    /* Main program length 30000 = C000 * 4 */

/*
   copy 0xC000 words from P1START to MPSTART,
   copy 0xC000 words from P1START+0xC000 to MPSTART+0x0xC000,
   copy 0xC000 words from P1START+0x18000 to MPSTART+0x18000.
*/

/* RAM Memory */
#define MPSTART    0x407000    /* Main program code RAM starting address */

.SEGMENT /PM seg_pmco;

.global _asm_cpy1;          /* _copy1 is defined here */

_asm_cpy1:
    leaf_entry;           /* entry macro 4-13 */
    i8=P1START;           /* pointer to Main program #1 */
    m8=1;                  /* M8 increments by 1 */
    l8=0;                  /* no circular buffer */
    i11=MPSTART;          /* load majority program at MPSTART */
    m11=1;                 /* M11 increments by 1 */
    l11=0;                 /* no circular buffer */
    lcntr=LEN, do mpcpy1 until lce;
    px=pm(i8,m8);          /* load instruction from Main program #1 */

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 50/212
File: AmicaSoftware.doc

```
    pm(i11,m11)=px;          /* write current instruction */
    nop;
    nop;
mpcpy1: nop;
    nop;
    nop;
    lcntr=LEN, do mpcpy2 until lce;
    px=pm(i8,m8);           /* load instruction from Main program #1 */
    pm(i11,m11)=px;        /* write current instruction */
    nop;
    nop;
mpcpy2: nop;
    nop;
    nop;
    lcntr=LEN, do mpcpy3 until lce;
    px=pm(i8,m8);           /* load instruction from Main program #1 */
    pm(i11,m11)=px;        /* write current instruction */
    nop;
    nop;
mpcpy3: nop;
    nop;
    nop;
    leaf_exit;             /* exit macro */
.ENDSEG;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 51/212
File: AmicaSoftware.doc

5.2.4. delflash.asm

```
//////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: delflash.asm  
//  
// Version: 1.0  
// Modified: 10/06/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: Flash write utility  
//  
//////////////////////////////////////  
  
#include <def21020.h>  
#include <asm_sprt.h>  
  
#define n 32768 /*number of words to be deleted */  
  
.SEGMENT /DM seg_dmda;  
    .var _asm_dle=0; /* asm_dle is defined here */  
    .global _asm_dle; /* needed so C can see it */  
.ENDSEG;  
  
.SEGMENT /PM seg_pmco;  
  
.global _asm_del; /* _asm_del is defined here */  
.extern _de_addr; /* destination address de_addr is defined in C file */  
  
_asm_del:  
    nop;  
    PMWAIT = b#00000010110001; /* imposta i wait states: 1ws bank 1, 4ws bank 0 */  
    DMWAIT = b#000011101111010010100101; /* DM 7ws b3, 7ws b2, 1ws b1, 1ws b0 */  
  
/* delete flash sector 0*/  
    r0=0x00aa00aa; /*write aa in 555*/  
    px1=r0;  
    px2=r0;  
    pm(0x555)=px;  
    r0=0x00550055; /*write 55 in 2aa*/  
    px1=r0;  
    px2=r0;  
    pm(0x2aa)=px;  
    r0=0x00800080; /*write 80 in 555*/  
    px1=r0;
```

```

px2=r0;
pm(0x555)=px;
r0=0x00aa00aa;    /*write aa in 555*/
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00550055;    /*write 55 in 2aa*/
px1=r0;
px2=r0;
pm(0x2aa)=px;
r0=0x00300030;    /*write 30 in the sector to be deleted*/
px1=r0;
px2=r0;
/*pm(0x000000)=px;*/
pm(_de_addr)=px;

/*idle; /* stop processor to avoid latchup*/

/* wait 10ms */
lcntr=0xffff, do rept until lce;
nop;
rept:  nop;
      nop;
      nop;
      nop;
      nop;

/* controllo fine cancellazione*/
cDQ7_0:  px=pm(0x000000);    /*check DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if sz jump cDQ5_0;
        jump  cDQ7_1; /*DQ7_0 is 1*/
cDQ5_0:  btst r0 by 5;
        if sz jump cDQ7_0;
        px=pm(0x000000);    /*DQ5_0 is 1, check again DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if not sz jump cDQ7_1; /*DQ7_0 is 1*/
        /*idle; /*error*/
        jump err;
cDQ7_1:  px=pm(0x000000);    /*check DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if sz jump cDQ5_1;
        jump  cDQ7_2; /*DQ7_1 is 1*/
cDQ5_1:  btst r1 by 5;
        if sz jump cDQ7_1;
        px=pm(0x000000);    /*DQ5_1 is 1, check again DQ7_1*/
        r1=px2;
        btst r1 by 7;

```

```
    if not sz jump cDQ7_2; /*DQ7_1 is 1*/
    /*idle; /*error*/
    jump err;
cDQ7_2:    px=pm(0x000000);    /*check DQ7_2*/
    r1=px2;
    btst r1 by 23;
    if sz jump cDQ5_2;
    jump    c_fine; /*DQ7_2 is 1*/
cDQ5_2:    btst r1 by 21;
    if sz jump cDQ7_2;
    px=pm(0x000000);    /*DQ5_2 is 1, check again DQ7_2*/
    r1=px2;
    btst r1 by 23;
    idle;
    if not sz jump c_fine; /*DQ7_2 is 1*/
    /*idle; /*error*/
    jump err;

c_fine: nop;
    nop;
    nop;
    nop;
    jump done;
    nop;
    nop;
    nop;

err:    nop;
    nop;
    nop;
    r0=0x1;
    dm(_asm_dle)=r0;
    nop;
    nop;
    nop;

done:  nop;
    nop;
    leaf_exit;    /* exit macro */

.ENDSEG;
```

5.2.5. utyflash.asm

```
////////////////////////////////////
//
// AMICA FOR AMS
//
// File: utyflash.asm
//
// Version: 1.0
// Modified: 08/06/07
// Author: F.Roncella
// Hardware: ADSP-21020
// Project: StarTracker
//
// Notes: Flash write utility
//
////////////////////////////////////

#include <def21020.h>
#include <asm_sprt.h>

#define n 32768 /*numero di parole da scrivere 32727*/

.SEGMENT /DM seg_dmda;
    .var _asm_wre=0; /* asm_wre is defined here */
    .global _asm_wre; /* needed so C can see it */
.ENDSEG;

.SEGMENT /PM seg_pmco;

.global _asm_wri; /* _asm_wri is defined here */
.extern _so_addr; /* source address so_addr is defined in C file */
.extern _de_addr; /* destination address de_addr is defined in C file */

_asm_wri:
    nop;
    PMWAIT = b#00000010110001; /* imposta i wait states: 1ws bank 1, 4ws bank 0 */
    DMWAIT = b#00001110111010010100101; /* DM 7ws b3, 7ws b2, 1ws b1, 1ws b0 */

/* cancella flash settore 0*/
    r0=0x00aa00aa; /*write aa in 555*/
    px1=r0;
    px2=r0;
    pm(0x555)=px;
    r0=0x00550055; /*write 55 in 2aa*/
    px1=r0;
    px2=r0;
    pm(0x2aa)=px;
    r0=0x00800080; /*write 80 in 555*/
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 55/212
File: AmicaSoftware.doc

```
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00aa00aa; /*write aa in 555*/
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00550055; /*write 55 in 2aa*/
px1=r0;
px2=r0;
pm(0x2aa)=px;
r0=0x00300030; /*write 30 in sector to be deleted*/
px1=r0;
px2=r0;
/*pm(0x000000)=px;*/
pm(_de_addr)=px;

/* wait 10ms */
lcnt=0xffff, do rept until lce;
nop;
rept: nop;
nop;
nop;
nop;

cDQ7_0: px=pm(0x000000); /*check DQ7_0*/
r0=px1;
btst r0 by 7;
if sz jump cDQ5_0;
jump cDQ7_1; /*DQ7_0 is 1*/
cDQ5_0: btst r0 by 5;
if sz jump cDQ7_0;
px=pm(0x000000); /*DQ5_0 is 1, check again DQ7_0*/
r0=px1;
btst r0 by 7;
if not sz jump cDQ7_1; /*DQ7_0 is 1*/
/*idle; /*error*/
jump err;
cDQ7_1: px=pm(0x000000); /*check DQ7_1*/
r1=px2;
btst r1 by 7;
if sz jump cDQ5_1;
jump cDQ7_2; /*DQ7_1 is 1*/
cDQ5_1: btst r1 by 5;
if sz jump cDQ7_1;
px=pm(0x000000); /*DQ5_1 is 1, check again DQ7_1*/
r1=px2;
btst r1 by 7;
if not sz jump cDQ7_2; /*DQ7_1 is 1*/
/*idle; /*error*/
```

```
        jump err;
cDQ7_2:  px=pm(0x000000); /*check DQ7_2*/
        r1=px2;
        btst r1 by 23;
        if sz jump cDQ5_2;
        jump c_fine; /*DQ7_2 is 1*/
cDQ5_2:  btst r1 by 21;
        if sz jump cDQ7_2;
        px=pm(0x000000); /*DQ5_2 is 1, check again DQ7_2*/
        r1=px2;
        btst r1 by 23;
        idle;
        if not sz jump c_fine; /*DQ7_2 is 1*/
        jump err;
c_fine: nop;
/* write data */
        b8=_so_addr;
        b9=_de_addr;
        l8=0; l9=0;
        m8=1; m9=-1;
        r0=0x00aa00aa; /*write aa in 555, unlock bypass*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;
        r0=0x00550055; /*write 55 in 2aa*/
        px1=r0;
        px2=r0;
        pm(0x2aa)=px;
        r0=0x00200020; /*write 20 in 555*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;
        lcntr=n, do scrivi until lce;
            r0=0x00A000A0; /*write A0 in 000000, unlock bypass program*/
            px1=r0;
            px2=r0;
            pm(0x000000)=px;
            px=pm(i8,m8); /*read data from ram*/
            r0=px1;
            r1=px2;
            pm(i9,m8)=px; /*write data*/
fcontr: px=pm(m9,i9); /*read data from flash*/
        r2=px1;
        r3=px2;
        comp(r0,r2);
        if ne jump fcontr;
        comp(r1,r3);
        if ne jump fcontr;
        nop;
        nop;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 57/212
File: AmicaSoftware.doc

```
                nop;
scrivi:  nop;
        r0=0x00900090;    /*write 90 in 000000, unlock bypass reset*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        r0=0x00000000;    /*write 00 in 000000*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        nop;
        nop;
        nop;
        jump done;
        nop;
        nop;
        nop;
err:     nop;
        nop;
        nop;
        r0=0x1;
        dm(_asm_wre)=r0;
        nop;
        nop;
        nop;
done:   nop;
        nop;
        leaf_exit;        /* exit macro */
.ENDSEG;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 58/212
File: AmicaSoftware.doc

5.2.6. wait.asm

```
//////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: wait.asm  
//  
// Version: 1.0  
// Modified: 27/05/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: Wait loop utility  
//  
//////////////////////////////////////  
  
/*#include <def21020.h>*/  
#include <asm_sprt.h>  
  
.SEGMENT /DM seg_dmda;  
  
.var _asm_lp=0; /* asm_lp is defined here */  
.global _asm_lp; /* needed so C can see it */  
  
.ENDSEG;  
  
.SEGMENT /PM seg_pmco;  
  
.global _asm_wait; /* _asm_wait is defined here */  
.extern _iwait; /* iwait is defined in C file */  
  
_asm_wait:  
leaf_entry; /* entry macro 4-13 */  
r8=dm(_iwait); /* access the global C variable */  
dm(_asm_lp)=r8; /* set asm variable loop to iwait */  
nop;  
lcntr=_asm_lp, do rept until lce;  
nop;  
rept: nop;  
nop;  
leaf_exit; /* exit macro */  
.ENDSEG;
```

5.2.7. wriflash.asm



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 59/212
File: AmicaSoftware.doc

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File:  wriflash.asm  
//  
// Version: 1.0  
// Modified: 10/06/07  
// Author:  F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes:  Flash write utility  
//  
////////////////////////////////////  
  
#include <def21020.h>  
#include <asm_sprt.h>  
  
#define      n          32768 /*numero di parole da scrivere 32727*/  
  
.SEGMENT /DM seg_dmda;  
    .var _asm_wre=0;      /* asm_wre is defined here */  
    .global _asm_wre;    /* needed so C can see it */  
.ENDSEG;  
  
.SEGMENT /PM seg_pmco;  
  
.global _asm_wri;        /* _asm_wri is defined here */  
.extern _so_addr;        /* source address so_addr is defined in C file */  
.extern _de_addr;        /* destination address de_addr is defined in C file */  
  
_asm_wri:  
    nop;  
    PMWAIT = b#00000010110001; /* imposta i wait states: 1ws banK 1, 4ws banK 0 */  
    DMWAIT = b#00001110111101001010101; /* DM 7ws b3, 7ws b2, 1ws b1, 1ws b0 */  
  
/* cancella flash settore 0*/  
    r0=0x00aa00aa;      /*write aa in 555*/  
    px1=r0;  
    px2=r0;  
    pm(0x555)=px;  
    r0=0x00550055;      /*write 55 in 2aa*/  
    px1=r0;  
    px2=r0;  
    pm(0x2aa)=px;  
    r0=0x00800080;      /*write 80 in 555*/  
    px1=r0;  
    px2=r0;  
    pm(0x555)=px;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 60/212
File: AmicaSoftware.doc

```
r0=0x00aa00aa;    /*write aa in 555*/
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00550055;    /*write 55 in 2aa*/
px1=r0;
px2=r0;
pm(0x2aa)=px;
r0=0x00300030;    /*write 30 in sector to be deleted*/
px1=r0;
px2=r0;
/*pm(0x000000)=px;*/
pm(_de_addr)=px;

/*idle; /*stop processor to avoid latchup*/

/* wait 10ms */
lcnt=0xffff, do rept until lce;
nop;
rept:  nop;
      nop;
      nop;
      nop;

/* check completed*/
cDQ7_0:  px=pm(0x000000);    /*check DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if sz jump cDQ5_0;
        jump  cDQ7_1; /*DQ7_0 is 1*/
cDQ5_0:  btst r0 by 5;
        if sz jump cDQ7_0;
        px=pm(0x000000);    /*DQ5_0 is 1, check again DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if not sz jump cDQ7_1; /*DQ7_0 is 1*/
        /*idle; /*error*/
        jump err;
cDQ7_1:  px=pm(0x000000);    /*check DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if sz jump cDQ5_1;
        jump  cDQ7_2; /*DQ7_1 is 1*/
cDQ5_1:  btst r1 by 5;
        if sz jump cDQ7_1;
        px=pm(0x000000);    /*DQ5_1 is 1, check again DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if not sz jump cDQ7_2; /*DQ7_1 is 1*/
        /*idle; /*error*/
```

```
        jump err;
cDQ7_2:  px=pm(0x000000);   /*check DQ7_2*/
        r1=px2;
        btst r1 by 23;
        if sz jump cDQ5_2;
        jump  c_fine; /*DQ7_2 is 1*/
cDQ5_2:  btst r1 by 21;
        if sz jump cDQ7_2;
        px=pm(0x000000);   /*DQ5_2 is 1, check again DQ7_2*/
        r1=px2;
        btst r1 by 23;
        idle;
        if not sz jump c_fine; /*DQ7_2 is 1*/
        /*idle; /*error*/
        jump err;

c_fine: nop;

/* write data */
        b8=_so_addr;
        b9=_de_addr;
        l8=0;  l9=0;
        m8=1; m9=-1;

        r0=0x00aa00aa;      /*write aa in 555, unlock bypass*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;
        r0=0x00550055;      /*write 55 in 2aa*/
        px1=r0;
        px2=r0;
        pm(0x2aa)=px;
        r0=0x00200020;      /*write 20 in 555*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;

        lctr=n, do scrivi until lce;
        r0=0x00A000A0;      /*write A0 in 000000, unlock bypass program*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        px=pm(i8,m8); /*read data from ram*/
        r0=px1;
        r1=px2;
        pm(i9,m8)=px; /*write data*/
fcontr: px=pm(m9,i9); /*read data from flash*/
        r2=px1;
        r3=px2;
        comp(r0,r2);
```

```
        if ne jump fcontr;
        comp(r1,r3);
        if ne jump fcontr;
        nop;
        nop;
        nop;
scrivi:  nop;

        r0=0x00900090;      /*write 90 in 000000, unlock bypass reset*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        r0=0x00000000;      /*write 00 in 000000*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        nop;
        nop;
        nop;
        jump done;
        nop;
        nop;
        nop;

err:    nop;
        nop;
        nop;
        r0=0x1;
        dm(_asm_wre)=r0;
        nop;
        nop;
        nop;

/*end*/

done:  nop;
        nop;
        leaf_exit;          /* exit macro */

.ENDSEG;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 63/212
File: AmicaSoftware.doc

5.2.8. loader.c

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: loader.c  
//  
// Version: 1.0  
// Modified: 10/06/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: Loader  
//  
// Notes: Loader application  
//  
////////////////////////////////////  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
  
#include <def21020.h> /* Bit definitions for the registers */  
  
#include "defs.h" /* Loader values and addresses definitions */  
  
#include <21020.h> /* Needed for set_flag */  
#include <signal.h> /* Needed for interrupt(), raise(), signal() */  
  
#define TRUE 1  
#define FALSE 0  
#define PAGE_DIM 0x8000  
#define BUF_LEN 512  
#define MAX_CMD 20  
#define WBUFLen 0x100  
#define PRG_LEN 0x100  
#define LWAIT 2000000L  
#define HKSTDBY 100000L  
  
////////////////////////////////////  
// global variables  
////////////////////////////////////  
  
int bFirst;  
int bTimerOn;  
int iSer1;  
long adc_ch[16];  
int hk_val[16];
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 64/212
File: AmicaSoftware.doc

```
int bmsg1std;
int bmsg2std;
int iNBytes1;
int iNBytes2;
unsigned char buf1[BUF_LEN];
unsigned char buf2[BUF_LEN];
int iBuf1In;
int iBuf2In;
unsigned char bufPrg[PRG_LEN];
int iBufPrIn;
unsigned char cmdque[MAX_CMD][BUF_LEN];
int iCmdIndx;
int bQueFull;

/* CRC16 utilities */
unsigned long crc_tab[256];
int computed=0;

int ivers=0; /* defined here as a global; used in ASM file as _ivers */
int iwait=0;
long so_addr=0;
long de_addr=0;
extern int asm_var; /* defined in ASM file as _asm_var */
extern int asm_lp;
extern int asm_wre;
extern int asm_dle;

////////////////////////////////////
// function prototypes
////////////////////////////////////

void init_addr(void);
void serialrx_ops(void);
int ReadAsyncHK(void);
int ReadHkChannel(long lAddress);
int std_loader(void);
void send_message(int iCode,int iPort);
void execute_program(void);
unsigned char GetSerChar(void);
int SendSerChar(unsigned char ch, int port);
int ReadSerStatusReg(int serial);
unsigned char ReadSerDataReg(int serial);
void WriteSerDataReg(unsigned char ch, int serial);
void ManageCmdQueue(void);
void delay_short(void);
void delay_long(void);
void wait(int idelay);
void make_crc_table(void);
unsigned long update_crc(unsigned long crc, volatile unsigned char *buf, int len);
unsigned long crc(volatile unsigned char *buf, int len);
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 65/212
File: AmicaSoftware.doc

```
int CheckMessageCrc(int iPort);

/* flash utilities */
int write_flash(long source_start_addr,long dest_start_addr);
int write_flash_from_buf(long dest_start_addr);
int delete_flash(long start_addr);
int addr_to_block(long addr);
long block_to_addr(int iblock);
long addr_to_startaddr(long addr);

/* assembly functions prototyped here */
void asm_cpy1(void);
void asm_copy(void);
void asm_wait(void);
void asm_wri(void);
void asm_del(void);

////////////////////////////////////
// void main()
////////////////////////////////////

void main(void)
{
    int enabled=0;
    int iRes;
    int lCount=0;
    int lHkCount=0;
    int iHkDtOk=FALSE;
    int iDefFail=FALSE;

    bFirst=TRUE;
    iBuf1In=0;
    iBuf2In=0;
    iCmdIndx=0;
    bQueFull=FALSE;
    bmsg1std=FALSE;
    bmsg2std=FALSE;

    make_crc_table();
    init_addr();

    // at boot, for safe operation, ensure cameras power off state:

    set_flag(SET_FLAG0,CLR_FLAG); /* camera 0 power off */
    set_flag(SET_FLAG1,CLR_FLAG); /* camera 1 power off */

    /* disable interrupts */
    // "BIT CLR IMASK 0x0000180;" -> 1 1000 0000 = bit7 & bit8 = IRQ1 & IRQ0

    asm("bit clr MODE1 0x1000; \
```

```
        nop; \  
        nop; \  
        nop;");  
  
// avoid interrupts use at boot; the Timer is substituted by a simple loop counter  
/*  
// set 10s timer and enable Timer interrupt  
// 20Mhz -> clock cycle 50ns -> 1s=20.000.000 cycles  
interrupt(SIG_TMZ, timer_handler);  
timer_set((unsigned int)200000000,(unsigned int)200000000);  
timer_on();  
*/  
  
bTimerOn=TRUE;  
  
// main loop  
  
while(1)  
{  
    // one counter check per cycle, and one serial check per cycle;  
    // one housekeeping data read out HKSTDBY cycles (100000=0.5s)  
  
    IHkCount++;  
  
    if(IHkCount>HKSTDBY)  
    {  
        iHkDtOk=ReadAsyncHK();  
        IHkCount=0;  
    }  
  
    if(bTimerOn)  
    {  
        ICount++;  
  
        if(ICount>LWAIT)  
        {  
            // timer (10s) expired before serial command reception:  
            // stop serial ports servicing and load default Main Program  
  
            bTimerOn=FALSE;  
  
            // load default Main Program  
  
            iRes = std_loader();  
  
            if(iRes>0)  
            {  
                // a good version (iRes) of Main Program has been loaded  
  
                send_message(0,0); // ok
```

```

// execute Main Program and exit
execute_program();
}
else
{
send_message(2,0); // CRC multiple error on default Main
Program

iDefFail=TRUE;

// continue serial ports servicing wait for commands
}
}
}
}
// serial ports servicing
serialrx_ops();

// if present, manage the oldest command in the queue
if(iCmdIndx>0)
{
// servicing one command in one main loop is enough
ManageCmdQueue();
}
}
// main loop end
}

void init_addr(void)
{
/* ok define in defs.h */
adc_ch[0]=adc1_ch0;
adc_ch[1]=adc1_ch1;
adc_ch[2]=adc1_ch2;
adc_ch[3]=adc1_ch3;
adc_ch[4]=adc1_ch4;
adc_ch[5]=adc1_ch5;
adc_ch[6]=adc1_ch6;
adc_ch[7]=adc1_ch7;
adc_ch[8]=adc2_ch0;
adc_ch[9]=adc2_ch1;
adc_ch[10]=adc2_ch2;
adc_ch[11]=adc2_ch3;
adc_ch[12]=adc2_ch4;

```

```
    adc_ch[13]=adc2_ch5;
    adc_ch[14]=adc2_ch6;
    adc_ch[15]=adc2_ch7;
}

void serialrx_ops(void)
{
    // SERIAL ports received data handler

    // ser0_data    0x80000000 // serial 0 data DM address
    // ser0_reg     0x80000005 // serial 0 register DM address
    // ser1_data    0x80000010 // serial 1 data DM address
    // ser1_reg     0x80000015 // serial 1 register DM address

    int msg1comp;
    int msg2comp;
    unsigned char cNewChar;

    msg1comp=FALSE;
    msg2comp=FALSE;

    // store received chars in buffer

    cNewChar=GetSerChar();

    if(cNewChar<0x100) // valid
    {
        if(iSer1)
            buf1[iBuf1In]=cNewChar;
        else
            buf2[iBuf2In]=cNewChar;
    }
    else
    {
        return;
    }

    if(iSer1)
    {
        if(!bmsg1std)
        {
            // initial condition; look for sync byte

            if(cNewChar==0xE5)
            {
                // sync byte: candidate start of new message
                bmsg1std=TRUE;
                iBuf1In++;
            }
        }
    }
}
```

```
else
{
    // message bytes

    if(iBuf1In==1)
    {
        // message nr_bytes
        iNBytes1=cNewChar;
    }

    iBuf1In++;

    if(iBuf1In==iNBytes1)
    {
        // message completed
        // prepare for the next message
        bmsg1std=FALSE;

        if(CheckMessageCrc(1))
        {
            // message consistency checked
            // enable message interpreter
            msg1comp=TRUE;
        }
    }
}

if(msg1comp) // add message to command queue
{
    if(iCmdIndx<MAX_CMD-1)
    {
        int i;
        for(i=0;i<iNBytes1;i++)
        {
            cmdque[iCmdIndx][i]=buf1[i];
        }
        iCmdIndx++; // increase queue index
    }
    else // queue full
    {
        bQueFull=TRUE;
    }
}
}
else
{
    if(!bmsg2std)
    {
        // initial condition; look for sync byte
```

```
        if(cNewChar==0xE5)
        {
            // sync byte: candidate start of new message
            bmsg2std=TRUE;
            iBuf2In++;
        }
    }
else
{
    // message bytes

    if(iBuf2In==1)
    {
        // message nr_bytes
        iNBytes2=cNewChar;
    }

    iBuf2In++;

    if(iBuf2In==iNBytes2)
    {
        // message completed
        // prepare for the next message
        bmsg2std=FALSE;

        if(CheckMessageCrc(2))
        {
            // message consistency checked
            // enable message interpreter
            msg2comp=TRUE;
        }
    }
}

if(msg2comp) // add message to command queue
{
    if(iCmdIdx<MAX_CMD-1)
    {
        int i;
        for(i=0;i<iNBytes2;i++)
        {
            cmdque[iCmdIdx][i]=buf2[i];
        }
        iCmdIdx++; // increase queue index
    }
else // queue full
{
    bQueFull=TRUE;
}
}
```

```
    }
    // exit; commands queue manager is at the end of the main loop
}

unsigned char GetSerChar(void)
{
    int value;
    unsigned char ch;

    // try serial 0

    value=ReadSerStatusReg(0);

    // if bit0 = 0 (RxRDY; receive data ready) change serial

    if((int)(value/2)*2==value) // RxRDY=0 -> read serial 1
    {
        value=ReadSerStatusReg(1);

        if((int)(value/2)*2==value) // RxRDY=0; no data to read
        {
            return 0xffff;
        }
        else // RxRDY=1; read received data
        {
            iSer1=0;
            delay_short();
            ch = ReadSerDataReg(1); // read data register ser1_data
            return ch;
        }
    }
    else // RxRDY=1; read received data
    {
        iSer1=1;
        delay_short();
        ch = ReadSerDataReg(0); // read data register ser0_data
        return ch;
    }

    return ch;
}

int SendSerChar(unsigned char ch, int port)
{
    // port can be 0 or 1
    // returns 1 if success; if 0 the caller is responsible of trying again

    int value;
    delay_long();
}
```

```
if(port==0)
{
    value=ReadSerStatusReg(0);

    // if bit5 = 0 try again

    value=(int)(value/32); // :32 -> rshift5

    if((int)(value/2)*2==value)
    {
        return 0;           // THRE=0; transmitter holding register empty
    }
    else
    {
        WriteSerDataReg(ch,0); // write data register ser0_data
        delay_short();
        return 1;           // done
    }
}
else if(port==1)
{
    value=ReadSerStatusReg(1);

    // if bit5 = 0 try again

    value=(int)(value/32); // :32 -> rshift5

    if((int)(value/2)*2==value)
    {
        return 0;           // LSB=0
    }
    else
    {
        WriteSerDataReg(ch,1); // write data register ser1_data
        delay_short();
        return 1;           // done
    }
}

return 0; // just in case of problems
}

int ReadSerStatusReg(int serial)
{
    // read status register ser0_reg or ser1_reg

    int value=0;

    if(serial==0)
    {
```

```
        asm( "%0=dm(0x80000005);"."=d" (value) ); // ser0_reg
    }
    else if(serial==1)
    {
        asm( "%0=dm(0x80000015);"."=d" (value) ); // ser1_reg
    }
    return value;
}

unsigned char ReadSerDataReg(int serial)
{
    // read data register ser0_data or ser1_data

    unsigned char ch;

    if(serial==0)
    {
        asm( "%0=dm(0x80000000);"."=d" (ch) ); // ser0_data
    }
    else if(serial==1)
    {
        asm( "%0=dm(0x80000010);"."=d" (ch) ); // ser1_data
    }
    return ch;
}

void WriteSerDataReg(unsigned char ch, int serial)
{
    // write data register ser0_data or ser1_data

    if(serial==0)
    {
        asm( "dm(0x80000000)=%0;"."=d" (ch) ); // ser0_data
    }
    else if(serial==1)
    {
        asm( "dm(0x80000010)=%0;"."=d" (ch) ); // ser1_data
    }
}

void ManageCmdQueue()
{
    // command messages interpreter; we are here because iCmdIndx>0

    int i,j;
    int iCmdId;
    iCmdId=-1;

    if(bFirst) // first time: kill auto-load Timer
    {
```

```
bFirst=FALSE;

// disable timer and continue serial ports servicing
bTimerOn=FALSE;

//timer_off();
//interrupt(SIG_TMZ, SIG_IGN);    // disable timer interrupt
//clear_interrupt(SIG_TMZ);      // Timer = 0 (low priority option)
}

// last command:
// cmdque[iCmdIdx-1][2] = lo byte of command ID word
// cmdque[iCmdIdx-1][3] = hi byte of command ID word

// oldest command:
// cmdque[0][2] = lo byte of command ID word
// cmdque[0][3] = hi byte of command ID word

iCmdId=(cmdque[0][3]*0x100) + cmdque[0][2];

switch(iCmdId)
{
case 1:
    {
        int iVersion;
        iVersion=cmdque[0][4];

        if((iVersion>0)&&(iVersion<3))
        {
            execute_program(iVersion);
        }
        else
        {
            // return error code
            send_message(5,0);
        }
    }
    break;

case 2:
    {
        int i;
        long lAddress;
        long len;
        long databuf[WBUFLLEN];
        lAddress=(((cmdque[0][6]*0x100) + cmdque[0][5])*0x100) + cmdque[0][4];
        len=(cmdque[0][1] - 11);

        for(i=0;i<len;i++)
        {
```

```
        databuf[i]=cmdque[0][i+9];
    }
    WriteMemory(databuf,len);
}
break;

default:
{
    send_message(4,0);
}
}

// first, shift queue: 1->0, 2->1, ... (iCmdIndex-1)->(iCmdIndex-2)
for(j=1;j<iCmdIndx;j++)
{
    for(i=0;i<BUF_LEN;i++)
    {
        cmdque[j-1][i]=cmdque[j][i];
    }
}

// second, decrease queue index
iCmdIndx--;
}

void delay_short()
{
    // ser_rit = 128 delay cycles for ram, 32 for flash
    wait(128);
}

void delay_long()
{
    // ser_rit*2 = 256 delay cycles for ram, 64 for flash
    wait(256);
}

void wait(int idelay)
{
    iwait=idelay;
    asm_wait();
}

int write_flash_from_buf(long dest_start_addr)
{
    // get data from standard buffer and write to flash.
    // if block number is known, call with: write_flash_from_buf(block_to_addr(page));

    int ret;
```

```
long source_start_addr;

source_start_addr=flashbuf;           // see defs.h
ret=write_flash(source_start_addr,dest_start_addr);
// ret 0=ok, 1=write error
return ret;
}

int write_flash(long source_start_addr,long dest_start_addr)
{
    // get data from arbitrary buffer and write to flash.
    // if block number is known, call with: write_flash(source_addr,block_to_addr(page));

    // so_addr defined here as a global; used in ASM file as _so_addr
    so_addr=source_start_addr;
    de_addr=dest_start_addr;
    asm_wri();
    // asm_wre 0=ok, 1=write error
    return asm_wre;
}

int delete_flash(long start_addr)
{
    // call: delete_flash(start_address);
    // or:      delete_flash(addr_to_startaddr(intermediate_address));
    // or:      delete_flash(block_to_addr(page));

    // de_addr defined here as a global; used in ASM file as _de_addr
    de_addr=start_addr;
    asm_del();
    // asm_dle 0=ok, 1=error
    return asm_dle;
}

int addr_to_block(long addr)
{
    // returns flash page 0..127

    int iblock;
    iblock = (int)(addr/PAGE_DIM);
    if((iblock<0)|| (iblock>127))
        iblock=127;
    return iblock;
}

long block_to_addr(int iblock)
{
    // flash page 0..127

    long addr;
```

```
    if((iblock<0)||iblock>127))
        iblock=127;
    addr=iblock*PAGE_DIM;
    return addr;
}

long addr_to_startaddr(long addr)
{
    int iblock;
    iblock = (int)(addr/PAGE_DIM);
    addr = block_to_addr(iblock);
    return addr;
}

int ReadAsyncHK(void)
{
    // read FPGA housekeeping data asynchronously

    int iExit=TRUE;

    // the HK data update rate is relatively low, so a data
    // read during a write operation is unlikely. Anyway, we
    // read each value two times, to point out such situation;
    // if the values are different, we read a third time (this
    // operation is much faster than HK values update).

    int value;
    int value1, value2;
    int iIndex=0;

    // read ADC values and store to global variables

    for(iIndex=0;iIndex<16;iIndex++)
    {
        value1=ReadHkChannel(adc_ch[iIndex]);
        value2=ReadHkChannel(adc_ch[iIndex]);
        if(value1==value2)
        {
            value=value1;
        }
        else
        {
            value=ReadHkChannel(adc_ch[iIndex]);
        }
        hk_val[iIndex]=value;
    }

    return iExit;
}
```

```
int ReadHkChannel(long lAddress)
{
    int value = 0;
    //asm( "%0=dm(adc1_ch0);":"=d"(value) );
    //asm( "%0=dm(%1);":"=d"(value):"d"(lAddress) );

    switch(lAddress)
    {
    case adc1_ch0:
        asm( "%0=dm(0x80000030);":"=d"(value) ); // adc1_ch0
        break;
    case adc1_ch1:
        asm( "%0=dm(0x80000031);":"=d"(value) ); // adc1_ch1
        break;
    case adc1_ch2:
        asm( "%0=dm(0x80000032);":"=d"(value) ); // adc1_ch2
        break;
    case adc1_ch3:
        asm( "%0=dm(0x80000033);":"=d"(value) ); // adc1_ch3
        break;
    case adc1_ch4:
        asm( "%0=dm(0x80000034);":"=d"(value) ); // adc1_ch4
        break;
    case adc1_ch5:
        asm( "%0=dm(0x80000035);":"=d"(value) ); // adc1_ch5
        break;
    case adc1_ch6:
        asm( "%0=dm(0x80000036);":"=d"(value) ); // adc1_ch6
        break;
    case adc1_ch7:
        asm( "%0=dm(0x80000037);":"=d"(value) ); // adc1_ch7
        break;
    case adc2_ch0:
        asm( "%0=dm(0x80000038);":"=d"(value) ); // adc2_ch0
        break;
    case adc2_ch1:
        asm( "%0=dm(0x80000039);":"=d"(value) ); // adc2_ch1
        break;
    case adc2_ch2:
        asm( "%0=dm(0x8000003A);":"=d"(value) ); // adc2_ch2
        break;
    case adc2_ch3:
        asm( "%0=dm(0x8000003B);":"=d"(value) ); // adc2_ch3
        break;
    case adc2_ch4:
        asm( "%0=dm(0x8000003C);":"=d"(value) ); // adc2_ch4
        break;
    case adc2_ch5:
        asm( "%0=dm(0x8000003D);":"=d"(value) ); // adc2_ch5
        break;
    }
```

```
case adc2_ch6:
    asm( "%0=dm(0x8000003E);"."=d"(value) ); // adc2_ch6
    break;
case adc2_ch7:
    asm( "%0=dm(0x8000003F);"."=d"(value) ); // adc2_ch7
    break;
}
return value;
}

int std_loader(void)
{
    // loop over main program versions (3,2,1), check integrity and load

    unsigned char ucCrcLo;
    unsigned char ucCrcHi;
    unsigned long CrcRead;
    unsigned long CrcCalc;
    int iResult;
    int iLoop;
    int iNumCopy;
    int bOkCrc;
    int value = 0;
    int bValid;

    iResult=0;
    iNumCopy=3;
    bOkCrc=FALSE;

    for(iLoop=iNumCopy;iLoop>0;iLoop--)
    {
        // first, check for program validation flag

        bValid=FALSE;

        // flg_prg2    0x00078000    // main program 3 pre-validation
        // flg_prg1    0x00040000    // main program 2 pre-validation
        // flg_prg0    0x00008000    // main program 1 pre-validation

        switch(iLoop)
        {
        case 3:
            asm( "%0=dm(0x00078000);"."=d" (value) ); // flg_prg2
            break;
        case 2:
            asm( "%0=dm(0x00040000);"."=d" (value) ); // flg_prg1
            break;
        case 1:
            asm( "%0=dm(0x00008000);"."=d" (value) ); // flg_prg0
            break;
        }
```

```
    }

    if(value==0xFFFFFFFF)
    {
        bValid=TRUE;
    }

    if(bValid)
    {
        iBufPrIn=0;

        ucCrcLo=bufPrg[iBufPrIn-1];
        ucCrcHi=bufPrg[iBufPrIn-2];
        CrcRead=ucCrcLo+(ucCrcHi*256);    // x256->lshiftx8
        CrcCalc = crc(bufPrg,iBufPrIn-1);

        if(CrcRead==CrcCalc)
            bOkCrc=TRUE;
        else
            bOkCrc=FALSE;

        bOkCrc=TRUE;

        if(bOkCrc)
        {
            // load and exit

            // copy program to RAM (starting at prg_addr=0x407000)
            // mprg_len=0x30000 locations from prg1_addr=0x10000 (or
prg2_addr=0x48000 or prg3_addr=0x80000)

            ivers=iLoop;
            asm_copy();    // call to assembly function; returns version in
asm_var

            // ivers=1,2,3
            if(asm_var==ivers)
            {
                iResult=ivers;
                break;
            }
        }
        else
        {
            if(iLoop==iNumCopy-1)
            {
                // Last copy not valid; wait for serial commands.
            }
        }
    }
}
```

```
    }
    return iResult;
}

void send_message(int iCode, int iPort)
{
    // iCode = 0    "ok"
    // iCode = 1    "generic error"
    // iCode = 2    "main program CRC check failed"
    // iCode = 3    "execute program index not valid"
    // iCode = 4    "received command id not valid"
    // iCode = 5    "program version requested not valid"
    // the list is incomplete

    int iInd;
    unsigned char msg[7];
    int iMsgLen=7;

    msg[0]=0xE5; // sync
    msg[1]=iMsgLen;// length
    msg[2]=0xFF; // id (lo)
    msg[3]=0x00; // id (hi)
    msg[4]=iCode; // code
    msg[5]=0x00; // crc
    msg[6]=0x00; // crc

    for(iInd=0;iInd<iMsgLen;iInd++)
    {
        SendSerChar(msg[iInd],iPort);
    }
}

void execute_program(void)
{
    // start execution of loaded program
    asm( "jump(0x00407100);");
}

int CheckMessageCrc(int iPort)
{
    unsigned char ucCrcLo;
    unsigned char ucCrcHi;
    unsigned long CrcRead;
    unsigned long CrcCalc;
    int iResult;

    if(iPort==1)
    {
        ucCrcLo=buf1[iBuf1In-1];
        ucCrcHi=buf1[iBuf1In-2];
    }
}
```

```

        CrcRead=ucCrcLo+(ucCrcHi*256);    // x256->lshiftx8

        // unsigned long crc(unsigned char *buf, int len);
        CrcCalc = crc(buf1,iBuf1In-1);
    }
    else
    {
        ucCrcLo=buf2[iBuf2In-1];
        ucCrcHi=buf2[iBuf2In-2];
        CrcRead=ucCrcLo+(ucCrcHi*256);    // x256->lshiftx8

        // unsigned long crc(unsigned char *buf, int len);
        CrcCalc = crc(buf2,iBuf2In-1);
    }

    if(CrcRead==CrcCalc)
        iResult=1;
    else
        iResult=0;

    return iResult;
}

void make_crc_table(void)
{
    // Build a table for a fast CRC calculation

    unsigned long c;
    int n, k;

    for(n = 0; n < 256; n++)
    {
        c = (unsigned long) n;
        for(k = 0; k < 8; k++)
        {
            if(c & 1)
                c = 0xedb88320L ^ (c >> 1);
            else
                c = c >> 1;
        }
        crc_tab[n] = c;
    }
    computed = 1;
}

unsigned long update_crc(unsigned long crc, volatile unsigned char *buf, int len)
{
    // Update a running CRC with the bytes buf[0..len-1].
    // The CRC should be initialized to all 1's, and the transmitted
    // value is the 1's complement of the final running CRC.

```

```
unsigned long c = crc;
int n;

if(!computed)
    make_crc_table();
for(n = 0; n < len; n++)
{
    c = crc_tab[(c ^ buf[n]) & 0xff] ^ (c >> 8);
}
return c;
}

unsigned long crc(volatile unsigned char *buf, int len)
{
    // Return the CRC of the bytes buf[0..len-1].

    return update_crc(0xffffffffL, buf, len) ^ 0xffffffffL;
}
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 84/212
File: AmicaSoftware.doc

5.2.9. def21020.h

```
/*
 *
 * def21020.h
 * STATUS REGISTER BIT DEFINITIONS FOR ADSP-21020.
 *
 * (c) Copyright 2001-2004 Analog Devices, Inc. All rights reserved.
 * $Revision: 1.1 $
 */
This include file contains a list of "defines" to enable the programmer to
use symbolic names for all of the system register bits for the ADSP-21020.
*/
#ifndef __DEF21020_H_
#define __DEF21020_H_

/* MODE1 register */
#define BR0 0x00000002 /* Bit 1: Bit-reverse for I0 (uses DMS0- only) */
#define SRCU 0x00000004 /* Bit 2: Alt. register select for comp. units */
#define SRD1H 0x00000008 /* Bit 3: DAG1 alt. register select (7-4) */
#define SRD1L 0x00000010 /* Bit 4: DAG1 alt. register select (3-0) */
#define SRD2H 0x00000020 /* Bit 5: DAG2 alt. register select (15-12) */
#define SRD2L 0x00000040 /* Bit 6: DAG2 alt. register select (11-8) */
#define SRRFH 0x00000080 /* Bit 7: Register file alt. select for R(15-8) */
#define SRRFL 0x00000400 /* Bit 10: Register file alt. select for R(7-0) */
#define NESTM 0x00000800 /* Bit 11: Interrupt nesting enable */
#define IRPTEN 0x00001000 /* Bit 12: Global interrupt enable */
#define ALUSAT 0x00002000 /* Bit 13: Enable ALU fixed-pt. saturation */
#define TRUNC 0x00008000 /* Bit 15: 1=fltg-pt. truncation 0=Rnd to nearest */
#define RND32 0x00010000 /* Bit 16: 1=32-bit fltg-pt.rounding 0=40-bit rnd */

/* MODE2 register */
#define IRQ0E 0x00000001 /* Bit 0: IRQ0- 1=edge sens. 0=level sens. */
#define IRQ1E 0x00000002 /* Bit 1: IRQ1- 1=edge sens. 0=level sens. */
#define IRQ2E 0x00000004 /* Bit 2: IRQ2- 1=edge sens. 0=level sens. */
#define IRQ3E 0x00000008 /* Bit 3: IRQ3- 1=edge sens. 0=level sens. */
#define CADIS 0x00000010 /* Bit 4: Cache disable */
#define TIMEN 0x00000020 /* Bit 5: Timer enable */
#define FLG0O 0x00008000 /* Bit 15: FLAG0 1=output 0=input */
#define FLG1O 0x00010000 /* Bit 16: FLAG1 1=output 0=input */
#define FLG2O 0x00020000 /* Bit 17: FLAG2 1=output 0=input */
#define FLG3O 0x00040000 /* Bit 18: FLAG3 1=output 0=input */
#define CAFRZ 0x00080000 /* Bit 19: Cache freeze */

/* ASTAT register */
#define AZ 0x00000001 /* Bit 0: ALU result zero or fltg-pt. underflow */
```

```

#define AV    0x00000002 /* Bit 1: ALU overflow */
#define AN    0x00000004 /* Bit 2: ALU result negative */
#define AC    0x00000008 /* Bit 3: ALU fixed-pt. carry */
#define AS    0x00000010 /* Bit 4: ALU X input sign (ABS and MANT ops) */
#define AI    0x00000020 /* Bit 5: ALU fltg-pt. invalid operation */
#define MN    0x00000040 /* Bit 6: Multiplier result negative */
#define MV    0x00000080 /* Bit 7: Multiplier overflow */
#define MU    0x00000100 /* Bit 8: Multiplier fltg-pt. underflow */
#define MI    0x00000200 /* Bit 9: Multiplier fltg-pt. invalid operation */
#define AF    0x00000400 /* Bit 10: ALU fltg-pt. operation */
#define SV    0x00000800 /* Bit 11: Shifter overflow */
#define SZ    0x00001000 /* Bit 12: Shifter result zero */
#define SS    0x00002000 /* Bit 13: Shifter input sign */
#define BTF   0x00040000 /* Bit 18: Bit test flag for system registers */
#define FLG0  0x00080000 /* Bit 19: FLAG0 value */
#define FLG1  0x00100000 /* Bit 20: FLAG1 value */
#define FLG2  0x00200000 /* Bit 21: FLAG2 value */
#define FLG3  0x00400000 /* Bit 22: FLAG3 value */
#define CACC0 0x01000000 /* Bit 24: Compare Accumulation Bit 0 */
#define CACC1 0x02000000 /* Bit 25: Compare Accumulation Bit 1 */
#define CACC2 0x04000000 /* Bit 26: Compare Accumulation Bit 2 */
#define CACC3 0x08000000 /* Bit 27: Compare Accumulation Bit 3 */
#define CACC4 0x10000000 /* Bit 28: Compare Accumulation Bit 4 */
#define CACC5 0x20000000 /* Bit 29: Compare Accumulation Bit 5 */
#define CACC6 0x40000000 /* Bit 30: Compare Accumulation Bit 6 */
#define CACC7 0x80000000 /* Bit 31: Compare Accumulation Bit 7 */

/* STKY register */
#define AUS   0x00000001 /* Bit 0: ALU fltg-pt. underflow */
#define AVS   0x00000002 /* Bit 1: ALU fltg-pt. overflow */
#define AOS   0x00000004 /* Bit 2: ALU fixed-pt. overflow */
#define AIS   0x00000020 /* Bit 5: ALU fltg-pt. invalid operation */
#define MOS   0x00000040 /* Bit 6: Multiplier fixed-pt. overflow */
#define MVS   0x00000080 /* Bit 7: Multiplier fltg-pt. overflow */
#define MUS   0x00000100 /* Bit 8: Multiplier fltg-pt. underflow */
#define MIS   0x00000200 /* Bit 9: Multiplier fltg-pt. invalid operation */
#define CB7S  0x00020000 /* Bit 17: DAG1 circular buffer 7 overflow */
#define CB15S 0x00040000 /* Bit 18: DAG2 circular buffer 15 overflow */
#define PCFL  0x00200000 /* Bit 21: PC stack full */
#define PCEM  0x00400000 /* Bit 22: PC stack empty */
#define SSOV  0x00800000 /* Bit 23: Status stack overflow (MODE1 and ASTAT) */
#define SSEM  0x01000000 /* Bit 24: Status stack empty */
#define LSOV  0x02000000 /* Bit 25: Loop stack overflow */
#define LSEM  0x04000000 /* Bit 26: Loop stack empty */

/* IRPTL and IMASK and IMASKP registers */
#define RSTI  0x00000002 /* Bit 1: Address: 08: Reset */
#define SOVFI 0x00000008 /* Bit 3: Address: 18: Stack overflow */
#define TMZHI 0x00000010 /* Bit 4: Address: 20: Timer = 0 (high priority) */

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 86/212
File: AmicaSoftware.doc

```
#define IRQ3I 0x00000020 /* Bit 5: Address: 28: IRQ3- asserted */
#define IRQ2I 0x00000040 /* Bit 6: Address: 30: IRQ2- asserted */
#define IRQ1I 0x00000080 /* Bit 7: Address: 38: IRQ1- asserted */
#define IRQ0I 0x00000100 /* Bit 8: Address: 40: IRQ0- asserted */
#define CB7I 0x00000800 /* Bit 11: Address: 58: Circ. buffer 7 overflow */
#define CB15I 0x00001000 /* Bit 12: Address: 60: Circ. buffer 15 overflow */
#define TMZLI 0x00004000 /* Bit 14: Address: 70: Timer = 0 (low priority) */
#define FIXI 0x00008000 /* Bit 15: Address: 78: Fixed-pt. overflow */
#define FLT0I 0x00010000 /* Bit 16: Address: 80: fltg-pt. overflow */
#define FLTUI 0x00020000 /* Bit 17: Address: 88: fltg-pt. underflow */
#define FLTI 0x00040000 /* Bit 18: Address: 90: fltg-pt. invalid */
#define SFT0I 0x01000000 /* Bit 24: Address: C0: user software int 0 */
#define SFT1I 0x02000000 /* Bit 25: Address: C8: user software int 1 */
#define SFT2I 0x04000000 /* Bit 26: Address: D0: user software int 2 */
#define SFT3I 0x08000000 /* Bit 27: Address: D8: user software int 3 */
#define SFT4I 0x10000000 /* Bit 28: Address: E0: user software int 4 */
#define SFT5I 0x20000000 /* Bit 29: Address: E8: user software int 5 */
#define SFT6I 0x40000000 /* Bit 30: Address: F0: user software int 6 */
#define SFT7I 0x80000000 /* Bit 31: Address: F8: user software int 7 */

#endif // __DEF21020_H_

/* end of file */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 87/212
File: AmicaSoftware.doc

5.2.10. defs.h

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File:  defs.h  
//  
// Version: 1.0  
// Modified: 30/05/07  
// Author:  F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes:  Loader code definitions  
//  
////////////////////////////////////  
  
#ifndef __DEFS_H_  
#define __DEFS_H_  
  
#define ser0_data      0x80000000 /* serial 0 data DM address */  
#define ser0_reg      0x80000005 /* serial 0 register DM address */  
#define ser1_data      0x80000010 /* serial 1 data DM address */  
#define ser1_reg      0x80000015 /* serial 1 register DM address */  
#define ser_rit 128 /* delay cycles: 128 for ram, 32 for flash */  
#define adc1_ch0      0x80000030 /* ADC1 channel 0 data address */  
#define adc1_ch1      0x80000031 /* ADC1 channel 1 data address */  
#define adc1_ch2      0x80000032 /* ADC1 channel 2 data address */  
#define adc1_ch3      0x80000033 /* ADC1 channel 3 data address */  
#define adc1_ch4      0x80000034 /* ADC1 channel 4 data address */  
#define adc1_ch5      0x80000035 /* ADC1 channel 5 data address */  
#define adc1_ch6      0x80000036 /* ADC1 channel 6 data address */  
#define adc1_ch7      0x80000037 /* ADC1 channel 7 data address */  
#define adc2_ch0      0x80000038 /* ADC2 channel 0 data address */  
#define adc2_ch1      0x80000039 /* ADC2 channel 1 data address */  
#define adc2_ch2      0x8000003A /* ADC2 channel 2 data address */  
#define adc2_ch3      0x8000003B /* ADC2 channel 3 data address */  
#define adc2_ch4      0x8000003C /* ADC2 channel 4 data address */  
#define adc2_ch5      0x8000003D /* ADC2 channel 5 data address */  
#define adc2_ch6      0x8000003E /* ADC2 channel 6 data address */  
#define adc2_ch7      0x8000003F /* ADC2 channel 7 data address */  
  
#define flg_prg0      0x008000 /* main program 1 validation */  
#define flg_prg1      0x040000 /* main program 2 validation */  
#define flg_prg2      0x078000 /* main program 3 validation */  
#define prg1_adr      0x010000 /* main program #1 start address */  
#define prg2_adr      0x048000 /* main program #2 start address */  
#define prg3_adr      0x080000 /* main program #3 start address */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 88/212
File: AmicaSoftware.doc

```
#define prg_addr      0x407000    /* main program init address */  
#define mprg_len     0x030000    /* main program length */  
#define prgstart 0x407100    /* main program execution start address */  
  
#define flashbuf     0x00018000  /* flash buffer start address */  
  
#endif // DEFS_H
```

5.3. Tracker

5.3.1. Interrupt.txt

```
// SIG Value      Description  
// -----  
// SIG_SOVF      Status stack or Loop stack overflow or PC stack full  
// SIG_TMZ0      Timer = 0 (high priority option)  
// SIG_IRQ3      Interrupt 3  
// SIG_IRQ2      Interrupt 2  
// SIG_IRQ1      Interrupt 1  
// SIG_IRQ0      Interrupt 0  
// SIG_CB7       Circular buffer 7 overflow  
// SIG_CB15      Circular buffer 15 overflow  
// SIG_TMZ       Timer = 0 (low priority option)  
// SIG_FIX       Fixed-point overflow  
// SIG_FLTO      Floating point overflow exception  
// SIG_FLTU      Floating point underflow exception  
// SIG_FLTI      Floating point invalid exception  
// SIG_USR0      User software interrupt 0  
// SIG_USR1      User software interrupt 1  
// SIG_USR2      User software interrupt 2  
// SIG_USR3      User software interrupt 3  
// SIG_USR4      User software interrupt 4  
// SIG_USR5      User software interrupt 5  
  
// Function Argument  Description  
// -----  
// SIG_DFL              The default action is taken.  
// SIG_IGN              The signal is ignored.  
// Function address    The function pointed to by func is executed.
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 89/212
File: AmicaSoftware.doc

5.3.2. tracker.ach

```
.system loader
.processor=ADSP21020;

!***** Program memory *****/
!vector table in flash
!.SEGMENT /ROM /BEGIN=0x000000 /END=0x0000FF /PM seg_rth;

!bootstrap program code in flash
.SEGMENT /ROM /BEGIN=0x000100 /END=0x0001FF /PM pm_bsco;
!loader program code in flash - three copies
.SEGMENT /ROM /BEGIN=0x000200 /END=0x0067FF /PM pm_ldco;

!main program code in flash - three copies
.SEGMENT /RAM /BEGIN=0x008000 /END=0x0ACFFF /PM pm_mpc;
!star catalogue in flash - two copies
.SEGMENT /RAM /BEGIN=0x0AD000 /END=0x14CFFF /PM pm_cats;

!loader vector table in ram
.SEGMENT /ROM /BEGIN=0x400000 /END=0x4000FF /PM seg_rth;
!loader program code in ram
.SEGMENT /RAM /BEGIN=0x400100 /END=0x4020FF /PM seg_pmco;
.SEGMENT /ROM /BEGIN=0x402100 /END=0x4021FF /PM seg_init;
.SEGMENT /RAM /BEGIN=0x402200 /END=0x402200 /PM pm_ldcnt;
.SEGMENT /RAM /BEGIN=0x402201 /END=0x402201 /PM seg_pmda;

!main program vector table in ram
.SEGMENT /ROM /BEGIN=0x407000 /END=0x4070FF /PM sg1_rth;
!main program program code in ram
.SEGMENT /RAM /BEGIN=0x407100 /END=0x435EFF /PM pm_mpram;
.SEGMENT /ROM /BEGIN=0x435F00 /END=0x435FFF /PM sg1_init;
.SEGMENT /RAM /BEGIN=0x436000 /END=0x436000 /PM sg1_pmda;

!***** Data memory *****/
!data ram a 40 bit
!.SEGMENT /RAM /BEGIN=0x00000000 /END=0x0003FFFF /DM dm_bank0;

.SEGMENT /RAM /BEGIN=0x00000000 /END=0x000039FF /DM seg_dmda;
.SEGMENT /RAM /BEGIN=0x00003A00 /END=0x000063FF /CSTACK /DM seg_stak;
.SEGMENT /RAM /BEGIN=0x00006400 /END=0x00008FFF /CHEAP /DM seg_heap;

.SEGMENT /RAM /BEGIN=0x0000F000 /END=0x000129FF /DM sg1_dmda;
.SEGMENT /RAM /BEGIN=0x00012A00 /END=0x000153FF /CSTACK /DM sg1_stak;
.SEGMENT /RAM /BEGIN=0x00015400 /END=0x00017FFF /CHEAP /DM sg1_heap;
.SEGMENT /RAM /BEGIN=0x00018000 /END=0x0001FFFF /DM seg_flbf;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 90/212
File: AmicaSoftware.doc

```
!data ram a 16 bit
.SEGMENT /RAM /BEGIN=0x20000000 /END=0x2003FFFF /DM dm_bank1;

!ram immagini
.SEGMENT /RAM /BEGIN=0x40000000 /END=0x4003FFFF /DM dm_bank2;

!porte i/o
.SEGMENT /RAM /BEGIN=0x80000000 /END=0x800000FF /DM dm_bank3;

.ENDSYS;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 91/212
File: AmicaSoftware.doc

5.3.3. delflash.asm

```
//////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: delflash.asm  
//  
// Version: 1.0  
// Modified: 10/06/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: Flash write utility  
//  
//////////////////////////////////////  
  
#include <def21020.h>  
#include <asm_sprt.h>  
  
#define n 32768 /*number of words to be deleted */  
  
.SEGMENT /DM seg_dmda;  
    .var _asm_dle=0; /* asm_dle is defined here */  
    .global _asm_dle; /* needed so C can see it */  
.ENDSEG;  
  
.SEGMENT /PM seg_pmco;  
  
.global _asm_del; /* _asm_del is defined here */  
.extern _de_addr; /* destination address de_addr is defined in C file */  
  
_asm_del:  
    nop;  
    PMWAIT = b#00000010110001; /* imposta i wait states: 1ws bank 1, 4ws bank 0 */  
    DMWAIT = b#000011101111010010100101; /* DM 7ws b3, 7ws b2, 1ws b1, 1ws b0 */  
  
/* delete flash sector 0*/  
    r0=0x00aa00aa; /*write aa in 555*/  
    px1=r0;  
    px2=r0;  
    pm(0x555)=px;  
    r0=0x00550055; /*write 55 in 2aa*/  
    px1=r0;  
    px2=r0;  
    pm(0x2aa)=px;  
    r0=0x00800080; /*write 80 in 555*/  
    px1=r0;
```

```
px2=r0;
pm(0x555)=px;
r0=0x00aa00aa;    /*write aa in 555*/
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00550055;    /*write 55 in 2aa*/
px1=r0;
px2=r0;
pm(0x2aa)=px;
r0=0x00300030;    /*write 30 in the sector to be deleted*/
px1=r0;
px2=r0;
/*pm(0x000000)=px;*/
pm(_de_addr)=px;

/*idle; /* stop processor to avoid latchup*/

/* wait 10ms */
lcntr=0xffff, do rept until lce;
nop;
rept:  nop;
      nop;
      nop;
      nop;
      nop;

/* controllo fine cancellazione*/
cDQ7_0:  px=pm(0x000000);    /*check DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if sz jump cDQ5_0;
        jump  cDQ7_1; /*DQ7_0 is 1*/
cDQ5_0:  btst r0 by 5;
        if sz jump cDQ7_0;
        px=pm(0x000000);    /*DQ5_0 is 1, check again DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if not sz jump cDQ7_1; /*DQ7_0 is 1*/
        /*idle; /*error*/
        jump err;
cDQ7_1:  px=pm(0x000000);    /*check DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if sz jump cDQ5_1;
        jump  cDQ7_2; /*DQ7_1 is 1*/
cDQ5_1:  btst r1 by 5;
        if sz jump cDQ7_1;
        px=pm(0x000000);    /*DQ5_1 is 1, check again DQ7_1*/
        r1=px2;
        btst r1 by 7;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 93/212
File: AmicaSoftware.doc

```
    if not sz jump cDQ7_2; /*DQ7_1 is 1*/
    /*idle; /*error*/
    jump err;
cDQ7_2:    px=pm(0x000000);    /*check DQ7_2*/
    r1=px2;
    btst r1 by 23;
    if sz jump cDQ5_2;
    jump    c_fine; /*DQ7_2 is 1*/
cDQ5_2:    btst r1 by 21;
    if sz jump cDQ7_2;
    px=pm(0x000000);    /*DQ5_2 is 1, check again DQ7_2*/
    r1=px2;
    btst r1 by 23;
    idle;
    if not sz jump c_fine; /*DQ7_2 is 1*/
    /*idle; /*error*/
    jump err;

c_fine: nop;
    nop;
    nop;
    nop;
    jump done;
    nop;
    nop;
    nop;

err:    nop;
    nop;
    nop;
    r0=0x1;
    dm(_asm_dle)=r0;
    nop;
    nop;
    nop;

done:  nop;
    nop;
    leaf_exit;    /* exit macro */

.ENDSEG;
```

5.3.4. utyflash.asm

```
////////////////////////////////////
//
// AMICA FOR AMS
//
// File: utyflash.asm
//
// Version: 1.0
// Modified: 08/06/07
// Author: F.Roncella
// Hardware: ADSP-21020
// Project: StarTracker
//
// Notes: Flash write utility
//
////////////////////////////////////

#include <def21020.h>
#include <asm_sprt.h>

#define n 32768 /*numero di parole da scrivere 32727*/

.SEGMENT /DM seg_dmda;
    .var _asm_wre=0; /* asm_wre is defined here */
    .global _asm_wre; /* needed so C can see it */
.ENDSEG;

.SEGMENT /PM seg_pmco;

.global _asm_wri; /* _asm_wri is defined here */
.extern _so_addr; /* source address so_addr is defined in C file */
.extern _de_addr; /* destination address de_addr is defined in C file */

_asm_wri:
    nop;
    PMWAIT = b#00000010110001; /* imposta i wait states: 1ws bank 1, 4ws bank 0 */
    DMWAIT = b#00001110111010010100101; /* DM 7ws b3, 7ws b2, 1ws b1, 1ws b0 */

/* cancella flash settore 0*/
    r0=0x00aa00aa; /*write aa in 555*/
    px1=r0;
    px2=r0;
    pm(0x555)=px;
    r0=0x00550055; /*write 55 in 2aa*/
    px1=r0;
    px2=r0;
    pm(0x2aa)=px;
    r0=0x00800080; /*write 80 in 555*/
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 95/212
File: AmicaSoftware.doc

```
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00aa00aa;    /*write aa in 555*/
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00550055;    /*write 55 in 2aa*/
px1=r0;
px2=r0;
pm(0x2aa)=px;
r0=0x00300030;    /*write 30 in sector to be deleted*/
px1=r0;
px2=r0;
/*pm(0x000000)=px;*/
pm(_de_addr)=px;

/* wait 10ms */
lcnt=0xffff, do rept until lce;
nop;
rept:  nop;
      nop;
      nop;
      nop;

cDQ7_0:  px=pm(0x000000);    /*check DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if sz jump cDQ5_0;
        jump  cDQ7_1; /*DQ7_0 is 1*/
cDQ5_0:  btst r0 by 5;
        if sz jump cDQ7_0;
        px=pm(0x000000);    /*DQ5_0 is 1, check again DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if not sz jump cDQ7_1; /*DQ7_0 is 1*/
        /*idle; /*error*/
        jump err;
cDQ7_1:  px=pm(0x000000);    /*check DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if sz jump cDQ5_1;
        jump  cDQ7_2; /*DQ7_1 is 1*/
cDQ5_1:  btst r1 by 5;
        if sz jump cDQ7_1;
        px=pm(0x000000);    /*DQ5_1 is 1, check again DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if not sz jump cDQ7_2; /*DQ7_1 is 1*/
        /*idle; /*error*/
```

```
        jump err;
cDQ7_2:  px=pm(0x000000);   /*check DQ7_2*/
        r1=px2;
        btst r1 by 23;
        if sz jump cDQ5_2;
        jump  c_fine; /*DQ7_2 is 1*/
cDQ5_2:  btst r1 by 21;
        if sz jump cDQ7_2;
        px=pm(0x000000);   /*DQ5_2 is 1, check again DQ7_2*/
        r1=px2;
        btst r1 by 23;
        idle;
        if not sz jump c_fine; /*DQ7_2 is 1*/
        jump err;
c_fine: nop;
/* write data */
        b8=_so_addr;
        b9=_de_addr;
        l8=0;  l9=0;
        m8=1;  m9=-1;
        r0=0x00aa00aa;     /*write aa in 555, unlock bypass*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;
        r0=0x00550055;     /*write 55 in 2aa*/
        px1=r0;
        px2=r0;
        pm(0x2aa)=px;
        r0=0x00200020;     /*write 20 in 555*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;
        lcntr=n, do scrivi until lce;
                r0=0x00A000A0;     /*write A0 in 000000, unlock bypass program*/
                px1=r0;
                px2=r0;
                pm(0x000000)=px;
                px=pm(i8,m8); /*read data from ram*/
                r0=px1;
                r1=px2;
                pm(i9,m8)=px; /*write data*/
fcontr: px=pm(m9,i9); /*read data from flash*/
        r2=px1;
        r3=px2;
        comp(r0,r2);
        if ne jump fcontr;
        comp(r1,r3);
        if ne jump fcontr;
        nop;
        nop;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 97/212
File: AmicaSoftware.doc

```
                nop;
scrivi:  nop;
        r0=0x00900090;    /*write 90 in 000000, unlock bypass reset*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        r0=0x00000000;    /*write 00 in 000000*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        nop;
        nop;
        nop;
        jump done;
        nop;
        nop;
        nop;
err:     nop;
        nop;
        nop;
        r0=0x1;
        dm(_asm_wre)=r0;
        nop;
        nop;
        nop;
done:   nop;
        nop;
        leaf_exit;        /* exit macro */
.ENDSEG;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 98/212
File: AmicaSoftware.doc

5.3.5. wait.asm

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: wait.asm  
//  
// Version: 1.0  
// Modified: 27/05/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: Wait loop utility  
//  
////////////////////////////////////  
  
/*#include <def21020.h>*/  
#include <asm_sprt.h>  
  
.SEGMENT /DM seg_dmda;  
  
.var _asm_lp=0; /* asm_lp is defined here */  
.global _asm_lp; /* needed so C can see it */  
  
.ENDSEG;  
  
.SEGMENT /PM seg_pmco;  
  
.global _asm_wait; /* _asm_wait is defined here */  
.extern _iwait; /* iwait is defined in C file */  
  
_asm_wait:  
leaf_entry; /* entry macro 4-13 */  
r8=dm(_iwait); /* access the global C variable */  
dm(_asm_lp)=r8; /* set asm variable loop to iwait */  
nop;  
lcntr=_asm_lp, do rept until lce;  
nop;  
rept: nop;  
nop;  
leaf_exit; /* exit macro */  
.ENDSEG;
```

5.3.6. wriflash.asm



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 99/212
File: AmicaSoftware.doc

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File:  wriflash.asm  
//  
// Version: 1.0  
// Modified: 10/06/07  
// Author:  F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes:  Flash write utility  
//  
////////////////////////////////////  
  
#include <def21020.h>  
#include <asm_sprt.h>  
  
#define      n          32768 /*numero di parole da scrivere 32727*/  
  
.SEGMENT /DM seg_dmda;  
    .var _asm_wre=0;      /* asm_wre is defined here */  
    .global _asm_wre;    /* needed so C can see it */  
.ENDSEG;  
  
.SEGMENT /PM seg_pmco;  
  
.global _asm_wri;        /* _asm_wri is defined here */  
.extern _so_addr;        /* source address so_addr is defined in C file */  
.extern _de_addr;        /* destination address de_addr is defined in C file */  
  
_asm_wri:  
    nop;  
    PMWAIT = b#00000010110001; /* imposta i wait states: 1ws bank 1, 4ws bank 0 */  
    DMWAIT = b#00001110111101001010101; /* DM 7ws b3, 7ws b2, 1ws b1, 1ws b0 */  
  
/* cancella flash settore 0*/  
    r0=0x00aa00aa;      /*write aa in 555*/  
    px1=r0;  
    px2=r0;  
    pm(0x555)=px;  
    r0=0x00550055;      /*write 55 in 2aa*/  
    px1=r0;  
    px2=r0;  
    pm(0x2aa)=px;  
    r0=0x00800080;      /*write 80 in 555*/  
    px1=r0;  
    px2=r0;  
    pm(0x555)=px;
```

```
r0=0x00aa00aa;    /*write aa in 555*/
px1=r0;
px2=r0;
pm(0x555)=px;
r0=0x00550055;    /*write 55 in 2aa*/
px1=r0;
px2=r0;
pm(0x2aa)=px;
r0=0x00300030;    /*write 30 in sector to be deleted*/
px1=r0;
px2=r0;
/*pm(0x000000)=px;*/
pm(_de_addr)=px;

/*idle; /*stop processor to avoid latchup*/

/* wait 10ms */
lcnt=0xffff, do rept until lce;
nop;
rept:  nop;
      nop;
      nop;
      nop;

/* check completed*/
cDQ7_0:  px=pm(0x000000);    /*check DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if sz jump cDQ5_0;
        jump  cDQ7_1; /*DQ7_0 is 1*/
cDQ5_0:  btst r0 by 5;
        if sz jump cDQ7_0;
        px=pm(0x000000);    /*DQ5_0 is 1, check again DQ7_0*/
        r0=px1;
        btst r0 by 7;
        if not sz jump cDQ7_1; /*DQ7_0 is 1*/
        /*idle; /*error*/
        jump err;
cDQ7_1:  px=pm(0x000000);    /*check DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if sz jump cDQ5_1;
        jump  cDQ7_2; /*DQ7_1 is 1*/
cDQ5_1:  btst r1 by 5;
        if sz jump cDQ7_1;
        px=pm(0x000000);    /*DQ5_1 is 1, check again DQ7_1*/
        r1=px2;
        btst r1 by 7;
        if not sz jump cDQ7_2; /*DQ7_1 is 1*/
        /*idle; /*error*/
```

```
        jump err;
cDQ7_2:  px=pm(0x000000);   /*check DQ7_2*/
        r1=px2;
        btst r1 by 23;
        if sz jump cDQ5_2;
        jump  c_fine; /*DQ7_2 is 1*/
cDQ5_2:  btst r1 by 21;
        if sz jump cDQ7_2;
        px=pm(0x000000);   /*DQ5_2 is 1, check again DQ7_2*/
        r1=px2;
        btst r1 by 23;
        idle;
        if not sz jump c_fine; /*DQ7_2 is 1*/
        /*idle; /*error*/
        jump err;

c_fine: nop;

/* write data */
        b8=_so_addr;
        b9=_de_addr;
        l8=0;  l9=0;
        m8=1; m9=-1;

        r0=0x00aa00aa;      /*write aa in 555, unlock bypass*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;
        r0=0x00550055;      /*write 55 in 2aa*/
        px1=r0;
        px2=r0;
        pm(0x2aa)=px;
        r0=0x00200020;      /*write 20 in 555*/
        px1=r0;
        px2=r0;
        pm(0x555)=px;

        lcntr=n, do scrivi until lce;
        r0=0x00A000A0;      /*write A0 in 000000, unlock bypass program*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        px=pm(i8,m8); /*read data from ram*/
        r0=px1;
        r1=px2;
        pm(i9,m8)=px; /*write data*/
fcontr: px=pm(m9,i9); /*read data from flash*/
        r2=px1;
        r3=px2;
        comp(r0,r2);
```

```
        if ne jump fcontr;
        comp(r1,r3);
        if ne jump fcontr;
        nop;
        nop;
        nop;
scrivi:  nop;

        r0=0x00900090;      /*write 90 in 000000, unlock bypass reset*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        r0=0x00000000;     /*write 00 in 000000*/
        px1=r0;
        px2=r0;
        pm(0x000000)=px;
        nop;
        nop;
        nop;
        jump done;
        nop;
        nop;
        nop;

err:    nop;
        nop;
        nop;
        r0=0x1;
        dm(_asm_wre)=r0;
        nop;
        nop;
        nop;

/*end*/

done:  nop;
        nop;
        leaf_exit;        /* exit macro */

.ENDSEG;
```

5.3.7. ImagePro.c

```
#include "ImagePro.h"

////////////////////////////////////
// 2D fit (calls MRQMIN, COVSRT, GAUSSJ, MRQCOF; MRQCOF calls model functions)
////////////////////////////////////

void fit(double x[], double y[], double sig[], int ndata, double a[], double iniguess[],
         int ia[], int ma, int *ok)
{
    // Marquardt fit 2D; least-squares non-linear fit with 'chi-square' minimization

    int i;
    int fit_ok;
    int iter;
    double **covar,**alpha,*chisq;
    double chisqnew,chisqold,lambda;

    covar=dmatrix(1,ma,1,ma);
    alpha=dmatrix(1,ma,1,ma);
    chisq=dvector(1,ma);

    lambda = -1.0;
    for(i=1;i<=ma;i++) a[i]=iniguess[i];
    //
    switch(flagpsf){
        case '1':           // gaussian
            mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgauss,&lambda,&fit_ok);
            break;
        case '2':           // gaussian with subpixels
            mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgausssub,&lambda,&fit_ok);
            break;
        case '3':           // Moffat
            mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fmoffat,&lambda,&fit_ok);
            break;
        case '4':           // Moffat with subpixels
            mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fmoffatsub,&lambda,&fit_ok);
            break;
        case '5':           // asymmetrical gaussian
            mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgaussasimm,&lambda,&fit_ok);

            break;
    }
    if (fit_ok==1)
    {
        *ok=fit_ok;
        free_dvector(chisq,1,ma);
        free_dmatrix(alpha,1,ma,1,ma);
    }
}
```

```
        free_dmatrix(covar,1,ma,1,ma);
        return;
    }
    iter=0;
    chisqnew = *chisq;
    chisqold = 0;
    do
    {
        do
        {
            chisqold=chisqnew;
            switch(flagpsf)
            {
                case '1':

                    mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgauss,&lambda,&fit_ok);
                    break;
                case '2':

                    mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgausssub,&lambda,&fit_ok);
                    break;
                case '3':

                    mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fmofffat,&lambda,&fit_ok);
                    break;
                case '4':

                    mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fmofffatsub,&lambda,&fit_ok);
                    break;
                case '5':

                    mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgaussasimm,&lambda,&fit_ok);
                    break;
            }
            if (fit_ok==1)
            {
                *ok=fit_ok;
                free_dvector(chisq,1,ma);
                free_dmatrix(alpha,1,ma,1,ma);
                free_dmatrix(covar,1,ma,1,ma);
                return;
            }
            chisqnew = *chisq;
            iter++;
            if(iter>=maxiter)
            {
                *ok=2;
                free_dvector(chisq,1,ma);
                free_dmatrix(alpha,1,ma,1,ma);
                free_dmatrix(covar,1,ma,1,ma);
```

```
        return;
    }
} while (chisqnew >=chisqold);
iter++;
if(iter>=maxiter)
{
    *ok=2;
    free_dvector(chisq,1,ma);
    free_dmatrix(alpha,1,ma,1,ma);
    free_dmatrix(covar,1,ma,1,ma);
    return;
}
} while ((chisqold-chisqnew)>0.01);

lambda = 0.0;
// last call
switch(flagpsf)
{
    case '1':
        mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgauss,&lambda,&fit_ok);
        break;
    case '2':
        mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgausssub,&lambda,&fit_ok);
        break;
    case '3':
        mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fmofffat,&lambda,&fit_ok);
        break;
    case '4':
        mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fmofffatsub,&lambda,&fit_ok);
        break;
    case '5':
        mrqmin(x,y,sig,ndata,a,ia,ma,covar,alpha,chisq,fgaussasimm,&lambda,&fit_ok);

        break;
}
if (fit_ok==1)
{
    *ok=fit_ok;
    free_dvector(chisq,1,ma);
    free_dmatrix(alpha,1,ma,1,ma);
    free_dmatrix(covar,1,ma,1,ma);
    return;
}

free_dvector(chisq,1,ma);
free_dmatrix(alpha,1,ma,1,ma);
free_dmatrix(covar,1,ma,1,ma);
}
```

#define NRANSI

```
void mrqmin(double x[], double y[], double sig[], int ndata, double a[], int ia[],
int ma, double **covar, double **alpha, double *chisq,
void (*funcs)(double, double [], double *, double [], int, int), double *alamda,
int *fitok)
{
int j,k,l,m;
int success;
static int mfit;
static double ochisq,*atry,*beta,*da,**oneda;

if (*alamda < 0.0)
{
atry=dvector(1,ma);
beta=dvector(1,ma);
da=dvector(1,ma);
for (mfit=0,j=1;j<=ma;j++)
if (ia[j]) mfit++;
oneda=dmatrix(1,mfit,1,1);
*alamda=0.001;
switch(flagpsf)
{
case '1':
mrqcof(x,y,sig,ndata,a,ia,ma,alpha,beta,chisq,fgauss);
break;
case '2':
mrqcof(x,y,sig,ndata,a,ia,ma,alpha,beta,chisq,fgausssub);
break;
case '3':
mrqcof(x,y,sig,ndata,a,ia,ma,alpha,beta,chisq,fmoffat);
break;
case '4':
mrqcof(x,y,sig,ndata,a,ia,ma,alpha,beta,chisq,fmoffatsub);
break;
case '5':
mrqcof(x,y,sig,ndata,a,ia,ma,alpha,beta,chisq,fgaussasimm);
break;
}
ochisq>(*chisq);
for (j=1;j<=ma;j++) atry[j]=a[j];
}
for (j=0,l=1;l<=ma;l++)
{
if (ia[l])
{
for (j++,k=0,m=1;m<=ma;m++)
{
if (ia[m])
{
k++;

```

```
                covar[j][k]=alpha[j][k];
            }
        }
        covar[j][j]=alpha[j][j]*(1.0+(*alamda));
        oneda[j][1]=beta[j];
    }
}
gaussj(covar,mfit,oneda,1,&success);
if (success==1)
{
    *fitok=success;
    free_dmatrix(oneda,1,mfit,1,1);
    free_dvector(da,1,ma);
    free_dvector(beta,1,ma);
    free_dvector(atry,1,ma);
    return;
}

for (j=1;j<=mfit;j++) da[j]=oneda[j][1];
if (*alamda == 0.0) {
    covsrt(covar,ma,ia,mfit);
    free_dmatrix(oneda,1,mfit,1,1);
    free_dvector(da,1,ma);
    free_dvector(beta,1,ma);
    free_dvector(atry,1,ma);
    return;
}
for (j=0,l=1;l<=ma;l++)
    if (ia[l]) atry[l]=a[l]+da[++j];
switch(flagpsf){
    case '1':
        mrqcof(x,y,sig,ndata,atry,ia,ma,covar,da,chisq,fgauss);
        break;
    case '2':
        mrqcof(x,y,sig,ndata,atry,ia,ma,covar,da,chisq,fgausssub);
        break;
    case '3':
        mrqcof(x,y,sig,ndata,atry,ia,ma,covar,da,chisq,fmoffat);
        break;
    case '4':
        mrqcof(x,y,sig,ndata,atry,ia,ma,covar,da,chisq,fmoffatsub);
        break;
    case '5':
        mrqcof(x,y,sig,ndata,atry,ia,ma,covar,da,chisq,fgaussasimm);
        break;
}
if (*chisq < ochisq)
{
    *alamda *= 0.1;
    ochisq=(*chisq);
}
```

```

        for (j=0,l=1;l<=ma;l++)
        {
            if (ia[l])
            {
                for (j++,k=0,m=1;m<=ma;m++)
                {
                    if (ia[m])
                    {
                        k++;
                        alpha[j][k]=covar[j][k];
                    }
                }
                beta[j]=da[j];
                a[l]=atry[l];
            }
        }
    }
else
{
    *alamda *= 10.0;
    *chisq=ochisq;
}
}
#undef NRANSI

#define SWAP(a,b) {swap=(a);(a)=(b);(b)=swap;}

void covsrt(double **covar, int ma, int ia[], int mfit)
{
    int i,j,k;
    double swap;

    for (i=mfit+1;i<=ma;i++)
        for (j=1;j<=i;j++) covar[i][j]=covar[j][i]=0.0;
    k=mfit;
    for (j=ma;j>=1;j--)
    {
        if (ia[j])
        {
            for (i=1;i<=ma;i++) SWAP(covar[i][k],covar[i][j])
            for (i=1;i<=ma;i++) SWAP(covar[k][i],covar[j][i])
            k--;
        }
    }
}
#undef SWAP

#define NRANSI
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;}

```

```

void gaussj(double **a, int n, double **b, int m, int *success)
{
    int *indxc,*indxr,*ipiv;
    int i,icol,irow,j,k,l,ll;
    double big,dum,pivinv,temp;

    indxc=ivector(1,n);
    indxr=ivector(1,n);
    ipiv=ivector(1,n);
    for (j=1;j<=n;j++) ipiv[j]=0;
    for (i=1;i<=n;i++)
    {
        big=0.0;
        for (j=1;j<=n;j++)
            if (ipiv[j] != 1)
                for (k=1;k<=n;k++)
                {
                    if (ipiv[k] == 0)
                    {
                        if (fabs(a[j][k]) >= big)
                        {
                            big=fabs(a[j][k]);
                            irow=j;
                            icol=k;
                        }
                    }
                    else if (ipiv[k] > 1)
                    {
                        *success=1;
                        free_ivector(ipiv,1,n);
                        free_ivector(indxr,1,n);
                        free_ivector(indxc,1,n);
                        return;
                    }
                }
        ++(ipiv[icol]);
        if (irow != icol)
        {
            for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
            for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
        }
        indxr[i]=irow;
        indxc[i]=icol;
        if (a[icol][icol] == 0.0)
        {
            *success=1;
            free_ivector(ipiv,1,n);
            free_ivector(indxr,1,n);
            free_ivector(indxc,1,n);
            return;
        }
    }
}

```

```

    }
    pivinv=1.0/a[icol][icol];
    a[icol][icol]=1.0;
    for (l=1;l<=n;l++) a[icol][l] *= pivinv;
    for (l=1;l<=m;l++) b[icol][l] *= pivinv;
    for (ll=1;ll<=n;ll++)
        if (ll != icol)
            {
                dum=a[ll][icol];
                a[ll][icol]=0.0;
                for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
                for (l=1;l<=m;l++) b[ll][l] -= b[icol][l]*dum;
            }
    }
    for (l=n;l>=1;l--)
    {
        if (indxr[l] != indxc[l])
            for (k=1;k<=n;k++)
                SWAP(a[k][indxr[l]],a[k][indxc[l]]);
    }
    free_ivector(ipiv,1,n);
    free_ivector(indxr,1,n);
    free_ivector(indxc,1,n);
}
#undef SWAP
#undef NRANSI

#define NRANSI

void mrqcof(double x[], double y[], double sig[], int ndata, double a[], int ia[],
    int ma, double **alpha, double beta[], double *chisq,
    void (*funcs)(double, double [], double *, double [], int, int))
{
    int i,j,k,l,m,mfit=0;
    double ymod,wt,sig2i,dy,*dyda;

    dyda=dvector(1,ma);
    for (j=1;j<=ma;j++)
        if (ia[j]) mfit++;
    for (j=1;j<=mfit;j++)
    {
        for (k=1;k<=j;k++) alpha[j][k]=0.0;
        beta[j]=0.0;
    }
    *chisq=0.0;
    for (i=1;i<=ndata;i++)
    {
        (*funcs)(x[i],a,&ymod,dyda,ma,colbox);
        sig2i=1.0/(sig[i]*sig[i]);
        dy=y[i]-ymod;

```

```

        for (j=0,l=1;l<=ma;l++)
        {
            if (ia[l])
            {
                wt=dyda[l]*sig2i;
                for (j++,k=0,m=1;m<=l;m++)
                    if (ia[m]) alpha[j][++k] += wt*dyda[m];
                beta[j] += dy*wt;
            }
        }
        *chisq += dy*dy*sig2i;
    }
    for (j=2;j<=mfit;j++)
        for (k=1;k<j;k++) alpha[k][j]=alpha[j][k];
    free_dvector(dyda,1,ma);
}
#undef NRANSI

void fgauss(double p, double a[], double *yfit, double dyda[], int ma, int col)
{
    int i,resto,x,y,pint;
    double psfk,num,den,espon,espon;

    pint = (int)p;
    resto = pint%col;
    if (resto==0) x = col;
        else x = resto;
    y = ((pint-x)/col)+1;
    *yfit=0.0;
    *yfit += (a[1]+(a[2]*x)+(a[3]*y));
    dyda[1]=1;
    dyda[2]=x;
    dyda[3]=y;
    for (i=4;i<ma;i+=4)
    {
        den = 2*a[i+3]*a[i+3];
        num = ((x-a[i+1])*(x-a[i+1])+(y-a[i+2])*(y-a[i+2]));
        espon = -num/den;
        espon = exp(espon);           // a[i]=Ik
        psfk = a[i]*espon;           // a[i+1]=Xk
        *yfit += psfk;               // a[i+2]=Yk
        dyda[i] = espon;             // a[i+3]=sigma_k
        dyda[i+1] = psfk*2*(x-a[i+1])/den;
        dyda[i+2] = psfk*2*(y-a[i+2])/den;
        dyda[i+3] = psfk*num/(a[i+3]*a[i+3]*a[i+3]);
    }
}

void fgaussub(double p, double a[], double *yfit, double dyda[], int ma, int col)
{

```

```

int i,u,v,resto,x,y,pint;
double psfk,num,den,esp,somm,subpix,numx,numy,sommxk,sommyk,sommsk,fattk;

pint = (int)p;
resto = pint%col;
if (resto==0) x = col;
    else x = resto;
y = ((pint-x)/col)+1;
*yfit=0.0;
*yfit += (a[1]+(a[2]*x)+(a[3]*y));
dyda[1]=1;
dyda[2]=x;
dyda[3]=y;
for (i=4;i<ma;i+=4)
{
    somm=0.0;
    sommxk=0.0;
    sommyk=0.0;
    sommsk=0.0;
    den = 2*a[i+3]*a[i+3];
    for(u=1;u<=4;u++)
    {
        for(v=1;v<=4;v++)
        {
            subpix=0.0;
            numx=x-0.625+0.25*u-a[i+1]; // a[i]=Ik
            numy=y-0.625+0.25*v-a[i+2]; // a[i+1]=Xk
            num=numx*numx + numy*numy; // a[i+2]=Yk
            esp = -num/den; //
            a[i+3]=sigma_k

            subpix = exp(esp);
            somm += subpix;
            sommxk += subpix*2*numx/den;
            sommyk += subpix*2*numy/den;
            sommsk += subpix*num/(a[i+3]*a[i+3]*a[i+3]);
        }
    }
    fattk=a[i]/16;
    psfk = fattk*somm;
    *yfit += psfk;
    dyda[i] = somm/16; //d(model)/d(Ik)
    dyda[i+1] =fattk*sommxk; //d(model)/d(Xk)
    dyda[i+2] = fattk*sommyk; //d(model)/d(Yk)
    dyda[i+3] = fattk*sommsk; //d(model)/d(sigma_k)
}
}

void f Moffat(double p, double a[], double *yfit, double dyda[], int ma, int col)
{
int i,resto,x,y,pint;

```

```

double psfk,den,base,esp1,esp2,term1,term2,term3;

pint = (int)p;
resto = pint%col;
if (resto==0) x = col;
    else x = resto;
y = ((pint-x)/col)+1;
*yfit=0.0;
*yfit += (a[1]+(a[2]*x)+(a[3]*y));
dyda[1]=1;
dyda[2]=x;
dyda[3]=y;
for (i=4;i<ma;i+=5) {
    den = a[i+3]*a[i+3];
    term1 = ((x-a[i+1])*(x-a[i+1])+(y-a[i+2])*(y-a[i+2])) / den;
    base = 1+term1;
    esp1 = -a[i+4];
    term2 = pow(base,esp1);
    psfk = a[i]*term2;
    *yfit += psfk;
    esp2 = esp1-1;
    term3 = 2*a[i+4]*a[i]*pow(base,esp2); // a[i+4]=tau_k
    dyda[i] = term2;
    dyda[i+1] = term3*(x-a[i+1])/den;
    dyda[i+2] = term3*(y-a[i+2])/den;
    dyda[i+3] = term3*term1/a[i+3];
    dyda[i+4] = -a[i]*term2*log(base);
}
}

void fmoftsub(double p, double a[], double *yfit, double dyda[], int ma, int col)
{
int i,u,v,resto,x,y,pint;
    double psfk,somm,subpix,numx,numy,num,den,base,esp,resp,gam,gamk,gamkm,fattm;

pint = (int)p;
resto = pint%col;
if (resto==0) x = col;
    else x = resto;
y = ((pint-x)/col)+1;
*yfit=0.0;
*yfit += (a[1]+(a[2]*x)+(a[3]*y));
dyda[1]=1;
dyda[2]=x;
dyda[3]=y;
for (i=4;i<ma;i+=5)
{
    somm = 0.0;
    den = a[i+3]*a[i+3];
    esp = -a[i+4];

```

```

        for(u=1;u<=4;u++)
        {
            for(v=1;v<=4;v++)
            {
                subpix = 0.0;
                numx = x-0.625+0.25*u-a[i+1];//      a[i]=Ik
                numy = y-0.625+0.25*v-a[i+2];//      a[i+1]=Xk
                num = numx*numx + numy*numy;        //      a[i+2]=Yk
                base = 1+(num/den);                //      a[i+3]=rho_k
                subpix = pow(base,esp);            //      a[i+4]=tau_k
                somm += subpix;
            }
        }
        psfk = somm*a[i]/16;
        *yfit += psfk;
        gamk = 0.0625*somm;
        resp = 1/esp;
        gam = pow(gamk,resp);
        gamkm = gamk/gam;
        fattm = 2*a[i]*a[i+4]*gamkm/(a[i+3]*a[i+3]);
        dyda[i] = gamk;
        dyda[i+1] = fattm*(x-a[i+1]);
        dyda[i+2] = fattm*(y-a[i+2]);
        dyda[i+3] = fattm*((x-a[i+1])*(x-a[i+1])+(y-a[i+2])*(y-a[i+2]))/a[i+3];
        dyda[i+4] = -a[i]*gamk*log(gam);
    }
}

void fgaussasimm(double p, double a[], double *yfit, double dyda[], int ma, int col)
{
    int i,resto,x,y,pint;
    double psfk,numx,numy,denx,deny,esp,espon;

    pint = (int)p;
    resto = pint%col;
    if (resto==0) x = col;
    else x = resto;
    y = ((pint-x)/col)+1;
    *yfit=0.0;
    *yfit += (a[1]+(a[2]*x)+(a[3]*y));
    dyda[1]=1;                //      a[i]=Ik
    dyda[2]=x;                //      a[i+1]=Xk
    dyda[3]=y;                //      a[i+2]=Yk
    for (i=4;i<ma;i+=5)
    {
        denx = 2*a[i+3]*a[i+3];        // a[i+3]=sigmax_k
        deny = 2*a[i+4]*a[i+4];        // a[i+4]=sigmay_k
        numx = (x-a[i+1])*(x-a[i+1]);
        numy = (y-a[i+2])*(y-a[i+2]);
        esp = -numx/denx -numy/deny;
    }
}

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 115/212
File: AmicaSoftware.doc

```
    espon = exp(esp);
    psfk = a[i]*espon;
    *yfit += psfk;
    dyda[i] = espon;
//d(model)/d(Ik)
    dyda[i+1] = psfk*2*(x-a[i+1])/denx; //d(model)/d(Xk)
    dyda[i+2] = psfk*2*(y-a[i+2])/deny; //d(model)/d(Yk)
    dyda[i+3] = psfk*numx/(a[i+3]*a[i+3]*a[i+3]); //d(model)/d(sigmamax_k)
    dyda[i+4] = psfk*numy/(a[i+4]*a[i+4]*a[i+4]); //d(model)/d(sigmay_k)
}
}

////////////////////////////////////
//      Simulated image
////////////////////////////////////

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
#define PI 3.1415927F

void ossimul(double **ossim, double **sigstim, void (*funcs)(double, double [], double *, double [], int, int))
{
    int i,j,pix;
    double ysim,*asim,*der;

    der=dvector(1,maossim);
    asim=dvector(1,maossim);

    parsim(asim);

    for(i=1;i<=nrowsim;i++)
    {
        for(j=1;j<=ncolsim;j++) ossim[i][j]=0.0;
    }
    for(i=1;i<=nrowsim;i++)
    {
        for(j=1;j<=ncolsim;j++)
        {
            pix=(ncolsim*(i-1))+j;
            (*funcs)(pix,asim,&ysim,der,maossim,ncolsim);
            ossim[i][j]+=ysim;
        }
    }
}
```

```
if (addnoise==1) noiseadd(ossim);

for(i=1;i<=nrowsim;i++)
{
    for(j=1;j<=ncolsim;j++)
    {
        sigsim[i][j]=1.0;
    }
}

void parsim(double *asim)
{
    int i;
    long seed,*idum;

    asim[1] = Bzero;
    asim[2] = Bx;
    asim[3] = By;

    seed=-1;
    idum=&seed;
    ran1(idum);

    for(i=4;i<maossim;i+=passo)
    {
        asim[i]=ran1(idum)*40000+10000;
        asim[i+1]=ran1(idum)*(ncolsim-39)+20;
        asim[i+2]=ran1(idum)*(nrowsim-39)+20;
        switch(modpsfsim)
        {
            case 1:
                asim[i+3]=sigmax;
                asim[i+4]=sigmay;
                break;
            case 2:
                asim[i+3]=sigma;
                break;
            case 3:
                asim[i+3]=rho;
                asim[i+4]=tau;
                break;
        }
    }
}

double ran1(long *idum)
{
    int j;
```

```
long k;
static long iy=0;
static long iv[NTAB];
double temp;

if (*idum <= 0 || !iy)
{
    if (-(*idum) < 1) *idum=1;
    else *idum = -(*idum);
    for (j=NTAB+7;j>=0;j--)
    {
        k=(*idum)/IQ;
        *idum=IA*( *idum-k*IQ)-IR*k;
        if (*idum < 0) *idum += IM;
        if (j < NTAB) iv[j] = *idum;
    }
    iy=iv[0];
}
k=(*idum)/IQ;
*idum=IA*( *idum-k*IQ)-IR*k;
if (*idum < 0) *idum += IM;
j=iy/NDIV;
iy=iv[j];
iv[j] = *idum;
if ((temp=AM*iy) > RNMX) return RNMX;
else return temp;
}
```

```
double gasdev(long *idum)
{
    static int iset=0;
    static double gset;
    double fac,rsq,v1,v2;

    if (iset == 0)
    {
        do
        {
            v1=2.0*ran1(idum)-1.0;
            v2=2.0*ran1(idum)-1.0;
            rsq=v1*v1+v2*v2;
        } while (rsq >= 1.0 || rsq == 0.0);
        fac=sqrt(-2.0*log(rsq)/rsq);
        gset=v1*fac;
        iset=1;
        return v2*fac;
    }
    else
    {
```

```
        iset=0;
        return gset;
    }
}

void noiseadd(double **ossim)
{
    int i,j;
    long seed,*idum;
    double noise;

    seed=-1;
    idum=&seed;
    ran1(idum);

    for(i=1;i<=nrowsim;i++)
    {
        for(j=1;j<=ncolsim;j++)
        {
            noise=fattmolt*gasdev(idum);
            ossim[i][j] += noise;
            if (ossim[i][j] < 0.0) ossim[i][j] = 0.0;
        }
    }
}

void addstar(double **ossim)
{
    int i,j,pix;
    double mag,magzero,intmax,yima,*param,*der;

    switch(modpsfsim)
    {
        case 1:
            param=dvector(1,8);
            der=dvector(1,8);
            param[7]=sigmax;
            param[8]=sigmay;
            break;

        case 2:
            param=dvector(1,7);
            der=dvector(1,7);
            param[7]=sigma;
            break;

        case 3:
            param=dvector(1,8);
            der=dvector(1,8);
            param[7]=rho;
            param[8]=tau;
            break;
    }
}
```

```
}
param[1]=0.0;      //      fondo (B0)
param[2]=0.0;      //      (Bx)
param[3]=0.0;      //      (By)
//param[4]=maxAdu; //      max intensità
param[5]=xnewstar; //      x centro
param[6]=ynewstar; //      y centro

if (maxormag==1)
{
    param[4]=maxAdu;
}
else
{
    mag = magnewstar;
    magzero=10.0;
    intmax=pow(10,(magzero-mag)/5)/sqrt(2*PI*sigma*sigma);
    param[4]=intmax;
}

switch(modpsfsim)
{
    case 1:
        for(i=1;i<=nrowsim;i++)
        {
            for(j=1;j<=ncolsim;j++)
            {
                pix=(ncolsim*(i-1))+j;
                fgaussasimm(pix,param,&yima,der,8,ncolsim);
                ossim[i][j]+=yima;
            }
        }
        break;
    case 2:
        for(i=1;i<=nrowsim;i++)
        {
            for(j=1;j<=ncolsim;j++)
            {
                pix=(ncolsim*(i-1))+j;
                fgauss(pix,param,&yima,der,7,ncolsim);
                ossim[i][j]+=yima;
            }
        }
        break;
    case 3:
        for(i=1;i<=nrowsim;i++)
        {
            for(j=1;j<=ncolsim;j++)
            {
                pix=(ncolsim*(i-1))+j;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 120/212
File: AmicaSoftware.doc

```
        fmoffat(pix,param,&yima,der,8,ncolsim);  
        ossim[i][j]+=yima;  
    }  
    }  
break;  
}  
}  
  
#undef IA  
#undef IM  
#undef AM  
#undef IQ  
#undef IR  
#undef NTAB  
#undef NDIV  
#undef EPS  
#undef RNMX  
  
#undef PI
```

5.3.8. marq.c

```
void OptionsFITMARQ()
{
    global_maxiter = 25;
    global_rigbox = 10;
    global_colbox = 10;
    global_numbrill = 5;
    global_iniguess1 = 1;
    global_iniguess2 = 3;

    if (psfmodel == 1) { // gauss asimmm
        global_psfmodel = 1;
        global_flagpsf = '5';
    }
    if (psfmodel == 2) { // gauss simmm
        global_psfmodel = 2;
        global_flagpsf = '1';
    }
    if (psfmodel == 3) { // gauss simmm sub
        global_psfmodel = 3;
        global_flagpsf = '2';
    }
    if (psfmodel == 4) { // moffat
        global_psfmodel = 4;
        global_flagpsf = '3';
    }
    if (psfmodel == 5) { // moffat sub
        global_psfmodel = 5;
        global_flagpsf = '4';
    }
}
```

5.3.9. nrutil2.c

```
// ANSI C utility file

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include "nrutil2.h"

#define NR_END 1
#define FREE_ARG char*

// standard error handler
void nrerror(char error_text[])
{
    fprintf(stderr, "run-time error...\n");
}
```

```
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"...now exiting to system...\n");
exit(1);
}

// allocate a float vector with subscript range v[nl..nh]
float *vector(long nl, long nh)
{
    float *v;
    v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl+NR_END;
}

// allocate an int vector with subscript range v[nl..nh]
int *ivector(long nl, long nh)
{
    int *v;
    v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl+NR_END;
}

// allocate an unsigned char vector with subscript range v[nl..nh]
unsigned char *cvector(long nl, long nh)
{
    unsigned char *v;
    v=(unsigned char *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(unsigned char)));
    if (!v) nrerror("allocation failure in cvector()");
    return v-nl+NR_END;
}

// allocate an unsigned long vector with subscript range v[nl..nh]
unsigned long *lvector(long nl, long nh)
{
    unsigned long *v;
    v=(unsigned long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
    if (!v) nrerror("allocation failure in lvector()");
    return v-nl+NR_END;
}

// allocate a double vector with subscript range v[nl..nh]
double *dvector(long nl, long nh)
{
    double *v;
    v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl+NR_END;
}
```

```
// allocate a float matrix with subscript range m[nrl..nrh][ncl..nch]
float **matrix(long nrl, long nrh, long ncl, long nch)
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    float **m;

    // allocate pointers to rows
    m=(float **) malloc((size_t)((nrow+NR_END)*sizeof(float*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    // allocate rows and set pointers to them
    m[nrl]=(float *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    // return pointer to array of pointers to rows
    return m;
}

// allocate a double matrix with subscript range m[nrl..nrh][ncl..nch]
double **dmatrix(long nrl, long nrh, long ncl, long nch)
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    double **m;

    // allocate pointers to rows
    m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    // allocate rows and set pointers to them
    m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
    if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    // return pointer to array of pointers to rows
    return m;
}

// allocate a int matrix with subscript range m[nrl..nrh][ncl..nch]
int **imatrix(long nrl, long nrh, long ncl, long nch)
```

```
{
    long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
    int **m;

    // allocate pointers to rows
    m=(int **) malloc((size_t)((nrow+NR_END)*sizeof(int*)));
    if (!m) perror("allocation failure 1 in matrix()");
    m += NR_END;
    m -= nrl;

    // allocate rows and set pointers to them
    m[nrl]=(int *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
    if (!m[nrl]) perror("allocation failure 2 in matrix()");
    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    // return pointer to array of pointers to rows
    return m;
}

// point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch]
float **submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch, long newrl, long newcl)
{
    long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
    float **m;

    // allocate array of pointers to rows
    m=(float **) malloc((size_t)((nrow+NR_END)*sizeof(float*)));
    if (!m) perror("allocation failure in submatrix()");
    m += NR_END;
    m -= newrl;

    // set pointers to rows
    for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

    // return pointer to array of pointers to rows
    return m;
}

// new
// point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch]
double **dsubmatrix(double **a, long oldrl, long oldrh, long oldcl, long oldch, long newrl, long newcl)
{
    long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
    double **m;

    // allocate array of pointers to rows
    m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
```

```

if (!m) nerror("allocation failure in submatrix()");
m += NR_END;
m -= newrl;

// set pointers to rows
for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

// return pointer to array of pointers to rows
return m;
}

/* allocate a float matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch)
{
    long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
    float **m;

    // allocate pointers to rows
    m=(float **) malloc((size_t)((nrow+NR_END)*sizeof(float*)));
    if (!m) nerror("allocation failure in convert_matrix()");
    m += NR_END;
    m -= nrl;

    // set pointers to rows
    m[nrl]=a-ncl;
    for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;

    // return pointer to array of pointers to rows
    return m;
}

// allocate a float 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh]
float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh)
{
    long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
    float ***t;

    // allocate pointers to pointers to rows
    t=(float ***) malloc((size_t)((nrow+NR_END)*sizeof(float**)));
    if (!t) nerror("allocation failure 1 in f3tensor()");
    t += NR_END;
    t -= nrl;

    // allocate pointers to rows and set pointers to them
    t[nrl]=(float **) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float*)));
    if (!t[nrl]) nerror("allocation failure 2 in f3tensor()");
    t[nrl] += NR_END;

```

```
t[nrl] -= ncl;

// allocate rows and set pointers to them
t[nrl][ncl]=(float *) malloc((size_t)((nrow*ncol*ndep+NR_END)*sizeof(float));
if (!t[nrl][ncl]) nerror("allocation failure 3 in f3tensor()");
t[nrl][ncl] += NR_END;
t[nrl][ncl] -= ndl;

for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
for(i=nrl+1;i<=nrh;i++) {
    t[i]=t[i-1]+ncol;
    t[i][ncl]=t[i-1][ncl]+ncol*ndep;
    for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
}

// return pointer to array of pointers to rows
return t;
}

// free a float vector allocated with vector()
void free_vector(float *v, long nl, long nh)
{
    free((FREE_ARG) (v+nl-NR_END));
}

// free an int vector allocated with ivector()
void free_ivector(int *v, long nl, long nh)
{
    free((FREE_ARG) (v+nl-NR_END));
}

// free an unsigned char vector allocated with cvector()
void free_cvector(unsigned char *v, long nl, long nh)
{
    free((FREE_ARG) (v+nl-NR_END));
}

// free an unsigned long vector allocated with lvector()
void free_lvector(unsigned long *v, long nl, long nh)
{
    free((FREE_ARG) (v+nl-NR_END));
}

// free a double vector allocated with dvector()
void free_dvector(double *v, long nl, long nh)
{
    free((FREE_ARG) (v+nl-NR_END));
}

// free a float matrix allocated by matrix()
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 127/212
File: AmicaSoftware.doc

```
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

// free a double matrix allocated by dmatrix()
void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch)
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

// free an int matrix allocated by imatrix()
void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)
{
    free((FREE_ARG) (m[nrl]+ncl-NR_END));
    free((FREE_ARG) (m+nrl-NR_END));
}

// free a submatrix allocated by submatrix()
void free_submatrix(float **b, long nrl, long nrh, long ncl, long nch)
{
    free((FREE_ARG) (b+nrl-NR_END));
}

// new
// free a submatrix allocated by dsubmatrix()
void free_dsubmatrix(double **b, long nrl, long nrh, long ncl, long nch)
{
    free((FREE_ARG) (b+nrl-NR_END));
}

// free a matrix allocated by convert_matrix()
void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch)
{
    free((FREE_ARG) (b+nrl-NR_END));
}

// free a float f3tensor allocated by f3tensor()
void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch, long ndl, long ndh)
{
    free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
    free((FREE_ARG) (t[nrl]+ncl-NR_END));
    free((FREE_ARG) (t+nrl-NR_END));
}
```

5.3.10. psfintegral.c

```
#include <math.h>
#include "psfintegral.h"

double gaussintegral(double max, double sigmax, double sigmay)
{
    // 20-points Gauss-Legendre integration
    return quad2d(funcgauss,-5*sigmax,5*sigmax,max,sigmax,sigmay);
    //      sigmax = 3 -> x1 = -15, x2 = 15
    //      returns volume integral
}

static double xsav;
static double (*nrfunc)(double, double, double, double, double);

double quad2d(double (*func)(double, double, double, double, double),
              double x1, double x2, double max, double sigmax, double sigmay)
{
    nrfunc=func;
    return qgaus(f1,x1,x2,max,sigmax,sigmay);
}

double f1(double x, double max, double sigmax, double sigmay)
{
    xsav=x;
    return qgaus(f2,yy1(x,sigmay),yy2(x,sigmay),max,sigmax,sigmay);
}

double f2(double y, double max, double sigmax, double sigmay)
{
    //      integrand f(x,y) evaluated for a fixed x
    return (*nrfunc)(xsav,y,max,sigmax,sigmay);
}

double yy1(double x, double sigmay)
{
    // ok rectangle
    return (-5*sigmay);
}

double yy2(double x, double sigmay)
{
    // ok rectangle
    return (5*sigmay);
}

double qgaus(double (*func)(double ,double ,double ,double), double a, double b, double max, double
sigmax, double sigmay)
```

```

{
    int j;
    double xr,xm,dx,s;
    static double x[21];
    static double w[21];

    gauleg(-1,1,x,w,20);

    xm=0.5*(b+a);
    xr=0.5*(b-a);
    s=0;
    for (j=1;j<=10;j++) {
        dx=xr*x[j];
        s += w[j]*((*func)(xm+dx,max,sigmax,sigmay)+(*func)(xm-dx,max,sigmax,sigmay));
    }
    return s *= xr;
}

#define EPS 3.0e-11

void gauleg(double est1, double est2, double x[], double w[], int n)
{
    int m,j,i;
    double z1,z,xm,xl,pp,p3,p2,p1;

    m=(n+1)/2;
    xm=0.5*(est2+est1);
    xl=0.5*(est2-est1);
    for (i=1;i<=m;i++) {
        z=cos(3.141592654*(i-0.25)/(n+0.5));
        do {
            p1=1.0;
            p2=0.0;
            for (j=1;j<=n;j++) {
                p3=p2;
                p2=p1;
                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
            }
            pp=n*(z*p1-p2)/(z*z-1.0);
            z1=z;
            z=z1-p1/pp;
        } while (fabs(z-z1) > EPS);
        x[i]=xm-xl*z;
        x[n+1-i]=xm+xl*z;
        w[i]=2.0*xl/((1.0-z*z)*pp*pp);
        w[n+1-i]=w[i];
    }
}
#undef EPS

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 130/212
File: AmicaSoftware.doc

```
double funcgauss(double x, double y, double max, double sigmax, double sigmay)
{
    //      2D function to be integrated (Gaussian)
    double val;
    val = max*exp(-(x*x)/(2*sigmax*sigmax)-(y*y)/(2*sigmay*sigmay));
    return val;
}
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 131/212
File: AmicaSoftware.doc

5.3.11. star_find.c

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: star_find.c  
//  
// Version: 1.0  
// Modified: 04/06/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: star search functions  
//  
//  
////////////////////////////////////  
  
#include "star_find.h"  
  
////////////////////////////////////  
// bool InitParams( void )  
////////////////////////////////////  
  
bool InitParams( void )  
{  
    // star search parameters init  
  
    bool bExit;  
    bExit = false;  
  
    // read noise (ADU) = 1.6  
    // gain (e-/ADU) = 54.0  
    // low good datum (sigmas) = 8.0  
    // high good datum (ADU) = 3300.0  
    // FWHM of object = 2.5  
    // threshold (sigmas) = 40.0  
    // sharpness low threshold = 0.0  
    // sharpness hi threshold = 1.1  
    // roundness low threshold = -2.0  
    // roundness hi threshold = 2.0  
  
    opt[0]=1.6;  
    opt[1]=54.0;  
    opt[2]=8.0;  
    opt[3]=3300.0;  
    opt[4]=2.5;  
    opt[5]=40.0;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 132/212
File: AmicaSoftware.doc

```
opt[6]=0.0;  
opt[7]=1.1;  
opt[8]=-2.0;  
opt[9]=2.0;  
  
// aperture radius 1 = 2.5  
// aperture radius 2 = 3.0  
// aperture radius 3 = 4.0  
// aperture radius 4 = 5.0  
// aperture radius 5 = 6.0  
// aperture radius 6 = 7.0  
// aperture radius 7 = 8.0  
// aperture radius 8 = 0.0  
// aperture radius 9 = 0.0  
// aperture radius 10 = 0.0  
// aperture radius 11 = 0.0  
// aperture radius 12 = 0.0  
// inner sky annulus radius = 10.0  
// outer sky annulus radius = 16.0
```

```
ap_r[0]=2.5;  
ap_r[1]=3.0;  
ap_r[2]=4.0;  
ap_r[3]=5.0;  
ap_r[4]=6.0;  
ap_r[5]=7.0;  
ap_r[6]=8.0;  
ap_r[7]=0.0;  
ap_r[8]=0.0;  
ap_r[9]=0.0;  
ap_r[10]=0.0;  
ap_r[11]=0.0;  
in_sky_r=10.0;  
out_sky_r=16.0;
```

```
bExit = true;
```

```
return bExit;
```

```
}
```

```
////////////////////////////////////  
// bool FullSearch( void )  
////////////////////////////////////
```

```
bool FullSearch( void )
```

```
{
```

```
    // (slow) search procedure on whole sky
```

```
    bool bExit;
```

```
    bExit = false;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 133/212
File: AmicaSoftware.doc

```
// start search engine
DaoMax(0);

// test
/*
int iRetDialog;
CString strQuestion;
strQuestion = "ok FULL search?";
iRetDialog = AfxMessageBox( strQuestion, MB_YESNO | MB_ICONQUESTION );
if( iRetDialog == IDNO )
{
    bExit=false;
}
else
{
    bExit=true;
}
Sleep(100);
*/

/*
free( stella );
free( hhh );
free( dx );
free( dy );
free( g );
free( skip );
free( s );
*/
return bExit;
}

////////////////////////////////////
// bool LocalSearch( void )
////////////////////////////////////

bool LocalSearch( void )
{
    // (fast) search around known coordinates

    // LocalSearch();
    // LocalSearch( [direction parameters] );

    bool bExit;
    bExit = false;

    // start search engine
    DaoMax(0);
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 134/212
File: AmicaSoftware.doc

```
// test
/*
int iRetDialog;
CString strQuestion;
strQuestion = "ok LOCAL search?";
iRetDialog = AfxMessageBox( strQuestion, MB_YESNO | MB_ICONQUESTION );
if( iRetDialog == IDNO )
{
    bExit=false;
}
else
{
    bExit=true;
}
Sleep(100);
*/

/*
free( stella );
free( hhh );
free( dx );
free( dy );
free( g );
free( skip );
free( s );
*/
return bExit;
}

/////////////////////////////////////////////////////////////////
// bool TrackStars( void )
/////////////////////////////////////////////////////////////////

bool TrackStars( void )
{
    // (fastest) refine known coordinates by known stars tracking
    // using image sub-squares

    bool bExit;
    bool bBorderApproaching;
    bExit = false;
    bBorderApproaching = false;

    DaoMax(1);

    //CheckFOV(void);
    bBorderApproaching = bCloseToBorder;

    // test
    /*
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 135/212
File: AmicaSoftware.doc

```
int iRetDialog;
CString strQuestion;
strQuestion = "ok TRACKING?";
iRetDialog = AfxMessageBox( strQuestion, MB_YESNO | MB_ICONQUESTION );
if( iRetDialog == IDNO )
{
    bExit=false;
}
else
{
    bExit=true;
}
Sleep(100);
*/

if(bBorderApproaching)
{
    // load entering stars (and kill exiting ones)

    // select 'iRegion' field in Star Catalogue

    // ...
}

return bExit;
}

////////////////////////////////////////////////////////////////
// bool InitTrackFunction( void )
////////////////////////////////////////////////////////////////

bool InitTrackFunction( void )
{
    // load values for star tracking function

    bool bExit;
    bExit = false;

    // ...

    return bExit;
}

////////////////////////////////////////////////////////////////
// Star centroids search. Faster than Stetson FIND (DAOPHOT);
// convolution calculated only over pixels above a threshold
// value. Output written on arrays; field recognition with mask.
// Instrumental magnitude calculated by PHOTSB.
////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////
// DAOMAX
////////////////////////////////////

void DaoMax(int iFunc)
{
    //iFunc = 0 full search; = 1 local search

    int             ore, minuti, secondi, nfiltro;
    char            nome[15]=" ", mask[20]=" ",
                  file1[15], file2[15],
                  str[80]=" ";
    unsigned int    i, j, l, k,
                  numero, estremo, taille,
                  maxph, maxpsf=100, colpar=64,
                  maxbox=13, maxsky=500,
                  nmask, numb, ciclo;
    float           x[50], y[50],
                  x1, sx1, y1, sy1,
                  *dx, *dy,
                  traslx, trasly;
    FILE            *fp, *fp1, *fp2, *fp3;
    int iarg=2;

    if(iarg!=2)
    {
        //printf("\n\n input file not valid");
    }
    else
    {
        stella = (MAG_APER *)malloc(2*sizeof(MAG_APER)*50);
        if(stella == NULL)
        {
            //printf("\n\n Unable to allocate frame memory STAR\n");
        }

        //hhh = (unsigned __int16 *)malloc(2*sizeof(*hhh)*(long)maxpic+34L);
        hhh = (int *)malloc(2*sizeof(*hhh)*(long)maxpic+34L);
        if(hhh == NULL)
        {
            //printf("\n\n Unable to allocate frame memory hhh\n");
        }

        dx = (float *)malloc(2*sizeof(*dx)*100L+34L);
        if(dx == NULL)
        {
            //printf("\n\n Unable to allocate frame memory dx\n");
        }

        dy = (float *)malloc(2*sizeof(*dy)*100L+34L);
```

```

        if(dy == NULL)
        {
            //printf("\n\n Unable to allocate frame memory dy\n");
        }

        g = (float *)malloc(2*sizeof(*g)*(long)(maxbox*maxbox)+34L);
        if(g == NULL)
        {
            //printf("\n\n Unable to allocate frame memory g\n");
        }

        //skip                =                (unsigned                __int16
*)malloc(2*sizeof(*skip)*(long)(maxbox*maxbox)+34L);
        skip = (int *)malloc(2*sizeof(*skip)*(long)(maxbox*maxbox)+34L);
        if(skip == NULL)
        {
            //printf("\n\n Unable to allocate frame memory skip\n");
        }

        s = (float *)malloc(2*sizeof(*s)*(long)maxsky+34L);
        if(s == NULL)
        {
            //printf("\n\n Unable to allocate frame memory s\n");
        }

        fp1=fopen("img.dat","r+t");
        if(fp1==NULL)
        {
            //printf("\n\n file not found");
        }
        else
        {
            while(fscanf(fp1,"%s %f %f %f",&data,&JD,&TU,&AirMass)!=EOF)
            {
                ore=(int)TU;
                minuti=(int)((TU-(float)ore)*60.);
                secondi=(int)((TU-(float)ore-((float)minuti)/60.)*3600.);

                //initgraph(&graphdriver,&graphmode,"");
                //setallpalette(&p);
                strcpy(nomefile,"");

                //rectangle(295,47,497,218);

                start();

                fscanf(fp1,"%s %s %s %f",&file1,&file2,&mask,&tempo);

                if(load(file1,colpar,0)==1)
                {

```

```
        //printf("\n\n image not found");
        goto exit;
    }
    filtro=file1[7];
    strcpy(titolo,file1);

    nfiltro=0;
    if(filtro=='B')nfiltro=11;
    if(filtro=='V')nfiltro=22;
    if(filtro=='R')nfiltro=33;
    if(filtro=='I')nfiltro=44;

    if(file2!=NULL)
    {
        if(load(file2,colpar,1)==1)
        {
            //printf("\n\n DARK image not found");
            goto exit;
        }
        strcat(titolo," - ");
        strcat(titolo,file2);
    }
    taille=strlen(mask)-4;
    strncat(nomefile,mask,taille);
    strcat(nomefile,".fot");

    //gotoxy(1,21);
    //printf("%s %s",titolo,nomefile);

    fp3=fopen("aper.fot","a+t");
    fprintf(fp3,"\n          FILE = %s ",titolo);
    fprintf(fp3,"\n %10s  TU=%8.5f  Tesp=%6.1fs  AirMass=%5.2f
filtro=%c \n",data,TU,tempo,AirMass,filtro);
    fclose(fp3);

    fp2=fopen(nomefile,"a+t");
    fprintf(fp2,"\n          %9.5f          %c%c%c%c          %c%c
%c%c",JD,data[0],data[1],data[2],data[3],data[5],data[6],data[8],data[9]);
    fprintf(fp2,"          %2d          %2d          %2d          %5.1f
%2d",ore,minuti,secondi,tempo,nfiltro);

    str[0]='\0';
    fp=fopen(mask,"rt");
    if(fp==NULL)
    {
        sprintf(str,"mask file %s not found",mask);
        report(str);
        do
        {
            opt[5]=opt[5]*0.8;
```

```

        if(opt[5]<opt[2])
        {
            report("0 stars found");
            goto exit;
        }
        //gotoxy(1,21);
        //printf("\n threshold = %6.2f ",opt[5]);
        nstar=0;
        if((maxfind(maxbox,maxsky)!=0)||(nstar>=50))
        {
            sprintf(str,"%d stars found",nstar);
            report(str);
            goto exit;
        }
        for(i=1;i<=nstar;i++)
        {
            sprintf((stella+i-1)->name,"Star(%02d)",i);
        }
    }while(nstar<1);
}
else
{
    nmask=0;
    while(fscanf(fp,"%f                                %f
%s",&x[nmask],&y[nmask],&nome)!=EOF)
    {
        strcpy((stella+nmask)->name,nome);
        nmask++;
    }
    extremo=1;
    do
    {
        nstar=0;

        if((maxfind(maxbox,maxsky)!=0)||(nstar>3*nmask)||(nstar>=50))
        {
            sprintf(str,"%s: %d stars found",mask,nstar);
            report(str);
            goto exit;
        }

        if((nstar<2)||(nstar<nmask/3))
        {
            opt[5]=opt[5]*0.8;
            if(opt[5]<opt[2])
            {
                sprintf(str,"%s:   min   threshold
reached; only %d stars found",mask,nstar);
                report(str);
            }
        }
    }
}

```

```

                                for(i=1;i<=nstar;i++)
                                {
                                printf((stella+i-1)-
>name,"Star(%02d)",i);
                                }
                                goto exit;
                                }
                                }
                                else
                                {
                                for(i=0;i<nmask;i++)
                                {
                                for(j=0;j<nstar;j++)
                                {
                                *(dx+i*nstar+j)=x[i]-
                                *(dy+i*nstar+j)=y[i]-
                                numero=0;
                                for(l=0;l<nmask;l++)
                                {
                                for(k=0;k<nstar;k++)
                                {
                                if((abs((int)(*(dx+i*nstar+j)-x[l]+centro_x[k]))<2)&&(abs((int)(*(dy+i*nstar+j)-
                                y[l]+centro_y[k]))<2))
                                {
                                numero++;
                                k=nstar;
                                }
                                }
                                }
                                if(numero>estremo)
                                {
                                estremo=numero;
                                traslx=*(dx+i*nstar+j);
                                trasly=*(dy+i*nstar+j);
                                }
                                }
                                }
                                }
                                if(estremo<max(2,nmask/3))
                                {
                                opt[5]=opt[5]*0.8;
                                if(opt[5]<opt[2])

```

```

                                {
                                sprintf(str,"mask %s compare failed;
%d stars found",mask,nstar);
                                report(str);
                                for(i=1;i<=nstar;i++)
                                {
                                sprintf((stella+i-1)-
>name,"Star(%02d)",i);
                                }
                                goto exit;
                                }
                                //gotoxy(1,21);
                                //printf("\n threshold = %6.2f ",opt[5]);
                                }
                                }while(estremo<max(2,nmask/3));

                                nstar=nmask;
                                for(i=0;i<nmask;i++)
                                {
                                sx1=x[i]-traslx;
                                sy1=y[i]-trasly;

                                if((sx1>6.)&&(sx1<187.)&&(sy1>5.)&&(sy1<160.))
                                {
                                centro((int)(sx1+0.5),(int)(sy1+0.5),i);
                                }
                                else
                                {
                                centro_x[i]=1000.0;
                                centro_y[i]=1000.0;
                                }
                                }
                                }

exit:                                ;

                                if (nstar>0)
                                {
                                maxph = maxpsf*maxpsf;
                                photsb(maxph);
                                for(i=0;i<nstar;i++)
                                {
                                fprintf(fp2,"                                %5.2f                                %5.2f                                ",(stella+i)-
>apmag,(stella+i)->magerr);
                                }
                                }
                                fclose(fp2);
                                fclose(fp);
                                strcpy(data,"");
                                //closegraph();

```

```
        }
    }
    fclose(fp1);
}

return;
}

////////////////////////////////////
// REPORT
////////////////////////////////////

int report(char msg[])
{
    FILE *fp;
    fp=fopen("report.fot","a+t");
    fprintf(fp,"\n%s\n",msg);
    fclose(fp);
    return 0;
}

////////////////////////////////////
// MAXFIND
////////////////////////////////////

int maxfind( unsigned int maxbox, unsigned int maxsky)
{
    float  pixels,radius,fwhm,sigsq,rsq,relerr,skylvl,temp,
           hmin,p,datum,height,denom,sgop,
           round,sharp,shrplo,shrpfi,rndlo,rndhi,
           sumg,sumgsq,sumgd,sumd,sg,sgsq,sgd,sd,wt,hx,hy,
           dgdx,sdgdx,sdgdxs,sddgdx,sgdgd,
           xcen,ycen,dx,dy,skymod;
    int    nhalf,nbox,middle,lastcl,lastro,jsq,nrows,
           i,j,k,n,ix,iy,jx,jy,kx,lx,ly,a,b,nx;

    hibad = opt[3];
    fwhm = opt[4];
    thresh = opt[5];
    shrplo = opt[6];
    shrpfi = opt[7];
    rndlo = opt[8];
    rndhi = opt[9];

    radius = amax1(2.001,0.637*fwhm);
    nhalf = min((maxbox-1)/2,(int)radius);
    nbox = 2*nhalf + 1;
    middle = nhalf;
    lastro = nrow - nhalf;
    lastcl = ncol - nhalf;
```

```
sigsq=(fwhm/2.35482)*(fwhm/2.35482);
radius = radius*radius;

sumg = 0.0;
sumgsq = 0.0;
pixels = 0.0;
for (j=0;j<nbox;j++)
{
    jsq = (j-middle)*(j-middle);
    for (i=0;i<nbox;i++)
    {
        rsq = (float)((i-middle)*(i-middle)+jsq);
        *(g+j*maxbox+i)=(float)(exp((double)(-0.5*rsq/sigsq)));
        if (rsq <= radius)
        {
            *(skip+j*maxbox+i) = 0;
            sumg = sumg + *(g+j*maxbox+i);
            sumgsq=sumgsq + (*(g+j*maxbox+i))*(*(g+j*maxbox+i));
            pixels = pixels + 1.0;
        }
        else
        {
            *(skip+j*maxbox+i) = 1;
        }
    }
}
if(pixels==0)
{
    //printf("\n pixels = 0");
    pixels=1;
}
denom=sumgsq-(sumg*sumg)/pixels;
sgop=sumg/pixels;
relerr=1.0/denom;
relerr=(float)(sqrt((double)relerr));

*(skip+middle*maxbox+middle)=1;
pixels = pixels-1.0;

//    in 'g' matrix (maxbox x maxbox) we put a Gaussian profile with
//    known FWHM. Significant values in subsequent calculations have
//    a value in 'skip' matrix; the number of these significant pixels
//    is in 'pixel' variable.

if(fsky(maxsky,&skymod)!=0)
{
    //printf("\n sky not found");
    //getch();
    return(1);
}
```

```

else
{
    //gotoxy(1,4);
    //printf("\nrelative error=%4.2f",relerr);

    readns = opt[0]*opt[0];
    phpadu = opt[1];
    hmin = (float)sqrt((double)(readns+amax1(0.0,skymod)/phpadu));
    lobad = skymod-(opt[2]*hmin);
    hmin = relerr*(opt[5]*hmin);
    readns = (float)sqrt((double)readns);

    nstar=0;

    for(jy=middle;jy<lastro;jy++)
    {
        for(jx=middle;jx<lastcl;jx++)
        {
            datum=(float)*(hhh+jy*ncol+jx);
            if(datum>(skymod+hmin))
            {
                for(iy=jy-nhalf;iy<=jy+nhalf;iy++)
                {
                    for(ix=jx-nhalf;ix<=jx+nhalf;ix++)
                    {
                        i=middle+(ix-jx);
                        j=middle+(iy-jy);
                        if(*(skip+j*maxbox+i) == 0)
                        {
                            if(datum<(float)*(hhh+iy*ncol+ix))
                            {
                                goto ttzz;
                            }
                        }
                    }
                }
            }
            // end of local maximum check

            *(skip+middle*maxbox+middle)=0;
            pixels = pixels+1.0;

            //      consider also central pixel for convolution

            sgd=0.;
            sd=0.;
            sgsq=sumgsq;
            sg=sumg;
            p=pixels;
            for(iy=jy-nhalf;iy<=jy+nhalf;iy++)
            {
                for(ix=jx-nhalf;ix<=jx+nhalf;ix++)

```

```

        {
            i=middle+(ix-jx);
            j=middle+(iy-jy);
            if(*(skip+j*maxbox+i) == 0)
            {
                datum=(float)*(hhh+iy*ncol+ix);
                if((datum>=lobad)&&(datum<=hibad))
                {
                    sgd=sgd+datum*(*(g+j*maxbox+i));
                    sd=sd+datum;
                }
                else
                {
                    sgsq=sgsq-
                    (*g+j*maxbox+i)*(*g+j*maxbox+i);
                    sg=sg-(*g+j*maxbox+i);
                    p=p-1.;
                }
            }
        }
    }
    if (p>1.5)
    {
        if(p<pixels)
        {
            sgsq=sgsq-(sg*sg)/p;
            if(sgsq!=0.)
            {
                sgd=(sgd-sg*sd/p)/sgsq;
            }
            else
            {
                sgd=0.;
            }
        }
        else
        {
            sgd=(sgd-sgop*sd)/denom;
        }
    }
    else
    {
        sgd=0.0;
    }

    height=sgd;
    *(skip+middle*maxbox+middle)=1;
    pixels = pixels-1.0;

```

```

//      reset central pixel of Gaussian profile

if(height<hmin)
{
    goto ttzz;
}          //      end of convolution check

sharp=0.;
datum=(float)*(hhh+jy*ncol+jx);
if((datum>lobad)&&(datum<hibad))
{
    p=0.;
    for(iy=jy-nhalf;iy<=jy+nhalf;iy++)
    {
        for(ix=jx-nhalf;ix<=jx+nhalf;ix++)
        {
            i=middle+(ix-jx);
            j=middle+(iy-jy);
            temp=0.0;
            if(*(skip+j*maxbox+i)==0)
            {
                datum=(float)*(hhh+iy*ncol+ix);

if((datum>=lobad)&&(datum<=hibad))
                {
                    temp=temp+(datum-
                    p=p+1.;
                }
            }
            sharp=sharp+temp;
        }
    }
    sharp=((float)*(hhh+jy*ncol+jx))-skymod-sharp/p)/height;
    if((sharp<shrplo)|| (sharp>shrphi))
    {
        goto ttzz;
    }
}          //      end of profile check

//      start centroid calculation

sumgd=0.0;
sumgsq=0.0;
sumg=0.0;
sumd=0.0;
sdgdx=0.0;
sdgdxs=0.0;
sddgdx=0.0;
sgdgd=0.0;

```

```

p=0.0;
n=0;
for(ix=jx-nhalf;ix<=jx+nhalf;ix++)
{
    i=middle+(ix-jx);
    sg=0.0;
    sd=0.0;
    for(iy=jy-nhalf;iy<=jy+nhalf;iy++)
    {
        j=middle+(iy-jy);
        wt=(float)(middle-abs(j-middle));
        datum=(float)*(hhh+iy*ncol+ix);
        if((datum>=lobad)&&(datum<=hibad))
        {
            sd=sd+(datum-skymod)*wt;
            sg=sg+wt*(*(g+j*maxbox+i));
        }
    }
    if(sg>0.0)
    {
        wt=(float)(middle-abs(i-middle));
        sumgd=sumgd+wt*sg*sd;
        sumgsq=sumgsq+wt*sg*sg;
        sumg=sumg+wt*sg;
        sumd=sumd+wt*sd;
        p=p+wt;
        n=n+1;
        dgdx=sg*(middle-i);
        sdgdxs=sdgdxs+wt*dgdx*dgdx;
        sdgdx=sdgdx+wt*dgdx;
        sddgdx=sddgdx+wt*sd*dgdx;
        sgdgdx=sgdgdx+wt*sg*dgdx;
    }
}
if(n<=2)
{
    goto ttzz;
}

hx=(sumgd-sumg*sumd/p)/(sumgsq-(sumg*sumg)/p);
if(hx<=0.0)
{
    goto ttzz;
}
skylvl=(sumd-hx*sumg)/p;
dx=(sgdgdx-(sddgdx-
sdgdx*(hx*sumg+skylvl*p)))/(hx*sdgdxs/sigsq);
xcen=jx+dx/(1.+(float)fabs((double)dx));

sumgd=0.0;

```

```
sumgsq=0.0;
sumg=0.0;
sumd=0.0;
sdgdx=0.0;
sdgdxs=0.0;
sddgdx=0.0;
sgdgdgdx=0.0;
p=0.0;
n=0;
for(iy=jy-nhalf;iy<=jy+nhalf;iy++)
{
    j=middle+(iy-jy);
    sg=0.0;
    sd=0.0;
    for(ix=jx-nhalf;ix<=jx+nhalf;ix++)
    {
        i=middle+(ix-jx);
        wt=(float)(middle-abs(i-middle));
        datum=(float)*(hhh+iy*ncol+ix);
        if((datum>=lobad)&&(datum<=hibad))
        {
            sd=sd+(datum-skymod)*wt;
            sg=sg+wt*(g+j*maxbox+i);
        }
    }
    if(sg>0.0)
    {
        wt=(float)(middle-abs(j-middle));
        sumgd=sumgd+wt*sg*sd;
        sumgsq=sumgsq+wt*sg*sg;
        sumg=sumg+wt*sg;
        sumd=sumd+wt*sd;
        p=p+wt;
        dgdx=sg*(middle-j);
        sdgdxs=sdgdxs+wt*dgdx*dgdx;
        sdgdx=sdgdx+wt*dgdx;
        sddgdx=sddgdx+wt*sd*dgdx;
        sgdgdgdx=sgdgdgdx+wt*sg*dgdx;
        n=n+1;
    }
}
if(n<=2)
{
    goto ttzz;
}
hy=(sumgd-sumg*sumd/p)/(sumgsq-(sumg*sumg)/p);
if(hy<=0.0)
{
    goto ttzz;
}
```

```

                                skylvl=(sumd-hy*sumg)/p;
                                dy=(sgdgd-(sddgd-
sdgd*(hy*sumg+skylvl*p)))/(hy*sdgd/sigsq);
                                ycen=iy+dy/(1.+(float)fabs((double)dy));

                                //      end of centroid calculation

                                round=2.*(hx-hy)/(hx+hy);
                                if((round<rndlo)||((round>rndhi))
                                {
                                        goto ttzz;
                                }      // end of round check

                                height=-2.5*(float)log10((double)(height/hmin));
                                centro_x[nstar]=xcen;
                                centro_y[nstar]=ycen;
                                nstar=nstar+1;
                                if(nstar==50)
                                        goto uscit;

                                //circle(300+(int)(xcen+0.5),50+(int)(ycen+0.5),6);

ttzz:      ;
                                }
                                }
uscit:;
                                }
                                return(0);
                                }
                                }

////////////////////////////////////
// PHOTSB
////////////////////////////////////

int photsb( int maxsky )
{
        double wt,costante,apmag[maxap+1],area[maxap+1],error[4];
        float  magerr[maxap+1],par[maxap+3],
                FWHM, sigma,
                skymod, skysig, skyskw, sigsq, skyvar,
                datum, r, rsq, fractn, edge, dum,
                xc, yc, dmag, apmxsq, round,
                rinsq, rout, routsq, dysq, edge1, edge2;

        int    i, j, k, l, naper, idum, lx, ly,
                istar, mx, my, nsky, nl;
                size_t  numb,larg=sizeof(s);

        FILE  *fp1;

        par[1]=ap_r[0];      // radius of aperture 1
        par[2]=ap_r[1];      // radius of aperture 2

```

```
par[3]=ap_r[2]; // radius of aperture 3
par[4]=ap_r[3]; // radius of aperture 4
par[5]=ap_r[4]; // radius of aperture 5
par[6]=ap_r[5]; // radius of aperture 6
par[7]=ap_r[6]; // radius of aperture 7
par[8]=ap_r[7]; // radius of aperture 8
par[9]=ap_r[8]; // radius of aperture 9
par[10]=ap_r[9]; // radius of aperture 10
par[11]=ap_r[10]; // radius of aperture 11
par[12]=ap_r[11]; // radius of aperture 12
par[13]=in_sky_r; // inner sky radius
par[14]=out_sky_r; // outer sky radius

fp1=fopen("aper.fot","a+t");
if(fp1==NULL)
{
    //printf("\n\n unable to open aper.fot file");
    //getch();
    return(1);
}

costante = 2.5*log10((double)tempo);

naper=maxap;
apmxsq=-1.0;
for(i=1;i<=maxap;i++)
{
    if (par[i]<=0.0){
        goto uzdz;}
    apmxsq=amax1(apmxsq,((par[i]+0.5)*(par[i]+0.5)));
}
goto uztz;
uzdz: ;
naper=i-1;
uztz: ;
rinsq=amax1(par[maxap+1],0.0);
rinsq=rinsq*rinsq;
routsq = ((float)maxsky)/PI + rinsq;
dum = par[maxap+2]*par[maxap+2];
if (dum>routsq)
{
    //printf("\n sky annulus specified too big");
    round=(float)sqrt((double)routsq);
    //printf("\n %f pixels is the largest outer sky radius permitted",round);
    //getch();
    return(1);
}
else
{
    if (dum<=rinsq)
```

```
        {
            //printf("\n outer sky radius not bigger than inner radius");
            //getch();
            return(2);
        }
    else
    {
        rout = par[maxap+2];
        routsq = dum;
    }
}

readns=opt[0];
readns=readns*readns;

istar=0;
dzzz: ;
istar=istar+1;
if(istar>nstar)
    goto nnzz;

xc=centro_x[istar-1];
yc=centro_y[istar-1];

if((xc>ncol)||(yc>nrow)||(xc<0.0)||(yc<0.0))
{
    (stella+istar-1)->apmag=99.99;
    (stella+istar-1)->magerr=9.99;
    goto dzzz;
}

lx = max(0,(int)(xc-rout));
mx = min(ncol,(int)(xc+rout));
ly = max(0,(int)(yc-rout));
my = min(nrow,(int)(yc+rout));
edge1=amin1(xc-0.5, ((float)ncol+0.5)-xc);
edge2=amin1(yc-0.5, ((float)nrow+0.5)-yc);
edge=amin1(edge1,edge2);

for(i=1;i<=naper;i++)
{
    apmag[i] = 0.0;
    if (edge<par[i]){
        apmag[i]=(-1.0E36);}
    area[i]=0.0;
}
nsky=0;
j=ly;
do
{
```

```

dysq=((float)j-yc)*((float)j-yc);
i=lx;
do
{
    rsq=dysq+((float)i-xc)*((float)i-xc);
    datum=(float)*(hhh+j*ncol+i);
    if((rsq<rinsq)||((rsq>routsq)) {
        goto duuz;}
    if((nsky>maxsky)||((datum<lobad)||((datum>hibad))){
        goto duuz;}
    nsky=nsky+1;
    *(s+nsky)=datum;
duuz:    ;

        if (rsq>apmxsq){
            goto dudc;}
        r=(float)sqrt((double)rsq)-0.5;
        k=1;
        do
        {
            if (r>par[k]){
                goto dudz;}
            fractn=amin1(1.0,(par[k]-r));
            fractn=amax1(0.0,fractn);
            if ((datum<lobad)||((datum>hibad))){
                apmag[k]=(double)(-1.0E36);}
            apmag[k] = apmag[k]+(double)(fractn*datum);
            area[k] = area[k]+(double)fractn;
dudz:    k=k+1;
        }
        while(k<=naper);
dudc:    ;

            i=i+1;
        }
        while (i<=mx);
        j=j+1;
    }
    while (j<=my);

    if (nsky<minsky)
    {
        //printf("\n not enough pixels in the sky annulus");
        //printf("\n check bad pixel thresholds; if ok we need a larger outer sky radius");
        //getch();
        return(3);
    }
    numb=nsky+1;
    qsort(s,numb,larg,(int(*))(const void *,const void *))rout1);

    if(mmm(s,nsky,&dum,&datum,&skymod,&skysig,&skyskw)!=0)

```

```

{
    //printf("\n mmm failed!");
    //getch();
    return(4);
}
skyvar=skysig*skysig;
sigsq=skyvar/(float)(nsky);

for(i=1;i<=naper;i++)
{
    if (skysig<0.){
        goto dduz;}
    apmag[i]=apmag[i]-(double)(skymod)*area[i];
    if (apmag[i]<=0.0){
        goto dduz;}
    error[1]=area[i]*(double)skyvar;
    error[2]=apmag[i]/(double)phpadu;
    error[3]=(double)sigsq*(area[i]*area[i]);

    magerr[i]=(float)(sqrt(error[1]+error[2]+error[3])/apmag[i]);
    magerr[i]=amin1(9.99,1.0857*magerr[i]);
    apmag[i]=costante-2.5*log10(apmag[i]);
    if (apmag[i] > (double)99.99){
        goto dduz;}

    seeing(xc,yc,skymod,&FWHM,&sigma);

    goto dddz;
dduz:    ;
        apmag[i]=99.99;
        magerr[i]=9.99;
        FWHM=99.99;
        sigma=9.99;
dddz:    ;
}

//circle(300+(int)(xc+0.5),50+(int)(yc+0.5),par[maxap+1]);
//circle(300+(int)(xc+0.5),50+(int)(yc+0.5),par[maxap+2]);

//gotoxy(1,10);
//printf(" star n.%3d      \n %s ",istar,(stella+istar-1)->name);
//gotoxy(1,13);
//printf(" xc=%5.2f yc=%5.2f      ",xc,yc);
//printf("\n FWHM=%4.2f%3.2f arcsec ",scalx*FWHM,scalx*sigma);
//printf("\n\n mag=%6.3f%5.3f \n sky=%6.3f ",apmag[2],magerr[2],skymod);

fprintf(fp1,"\nObject Name = %s",(stella+istar-1)->name);
fprintf(fp1,"\ncenter=(%5.2f,%5.2f) * FWHM=%4.2f%3.2f * sky=%5.1f%3.1f *
skew=%4.2f\n",xc,yc,FWHM,sigma,skymod,amin1(99.99,skysig),amin1(99.99,amax1(-99.99,skyskw)));
for(k=1;k<=naper;k++)

```

```
{
    fprintf(fp1," r = %3.1f ",par[k]);
}
fprintf(fp1,"\n");
for(k=1;k<=naper;k++)
{
    fprintf(fp1,"%5.2f\n%4.2f",apmag[k],magerr[k]);
}
fprintf(fp1,"\n");

(stella+istar-1)->apmag=apmag[2];
(stella+istar-1)->magerr=magerr[2];

goto dzzz;

nnzz: ;
if(!fclose(fp1))
    return(0);
else
{
    //printf("unable to close file");
    //getch();
    return(1);
}
}

////////////////////////////////////
// FSKY
////////////////////////////////////

int fsky( unsigned int max, float *smod )
{
    // Sky background determination

    int          istep,lx,ly,nox,n,irow,i,ifirst,j;
    float        skymn,skymed,skymod,skysig,skyskw;
    size_t       larg=sizeof(s),numb;

    istep = maxpic/max + 1;
    ifirst = 0;
    n = 0;
    irow=0;
    do
    {
        irow = irow+1;
        ifirst = ifirst+1;
        if(ifirst>istep)
            ifirst=ifirst-istep;
        i = ifirst-1;
```

```

        j = irow-1;
        do
        {
            if((float)*(hhh+j*ncol+i) > hibad)
            {
                i = i+1;
            }
            else
            {
                n = n+1;
                *(s+n) = (float)*(hhh+j*ncol+i);
                if (n==max)
                {
                    i=ncol+1;
                    irow=nrow;
                }
                else
                {
                    i=i+istep;
                }
            }
        }
        while (i <= ncol);
    }
    while (irow < nrow);
    numb=n+1;
    qsort(s,numb,larg,(int(*))(const void *,const void *))rout1);
    if(mmm(s,n,&skymn,&skymed,&skymod,&skysig,&skyskw)!=0)
    {
        //printf("\n mmm failed");
        //getch();
        return(1);
    }

    //gotoxy(1,2);
    //printf("\n skymod = %6.3f  \n skysig = %6.3f  ",skymod,skysig);

    *smod=skymod;
    return(0);
}

////////////////////////////////////
// MMM
////////////////////////////////////

int mmm(float *sky,unsigned int nsky,float *skymn,float *skymed,float *skymod,float *sigma,float *skew)
{
    int    minimm,maximm,niter,istep,jstep,redo,
           i,maxit=30,a,b;
    float  cut,cut1,cut2,delta,skymid,r,sign,x,center,side;

```

```

double sum,sumsq;

if(nsky<=0)
    goto nnzz;
skymid=0.5*(*(sky+(nsky+1)/2)+*(sky+(nsky/2)+1));
sum=(double)0.0;
sumsq=(double)0.0;
cut1=amin1(*(sky+nsky)-skymid,hibad-skymid);
cut1=amin1(skymid-*(sky+1),cut1);
cut2=skymid+cut1;
cut1=skymid-cut1;
minimm=0;
for(i=1;i<=nsky;i++)
{
    if (*(sky+i)<cut1)
    {
        minimm=i;
        goto uzuz;
    }
    if (*(sky+i)>cut2)
        goto uzdz;
    delta=*(sky+i)-skymid;
    sum=sum+(double)delta;
    sumsq=sumsq+(double)(delta*delta);
    maximm=i;
uzuz: ;
}

uzdz: *skymed=0.5*(*(sky+(minimm+maximm+1)/2)+*(sky+((minimm+maximm)/2)+1));
*skymn=(float)(sum/(double)(maximm-minimm));
*sigma=(float)(sqrt(sumsq/(double)(maximm-minimm)-
(double)((*skymn)*(*skymn))));
*skymn=*skymn+skymid;
*skymod=*skymn;
if (*skymed<*skymn)
    *skymod=3.*(*skymed)-2.*(*skymn);

niter=0;
dzzz: ;
niter=niter+1;
if((niter>maxit)||((maximm-minimm)<minsky))
    goto nnzz;
r=(float)log10((double)(maximm-minimm));
r=amax1(2.,(-0.1042*r+1.1695)*r+0.8895);
cut=r*(sigma)+0.5*fabs(*skymn-*skymod);
cut=amax1(1.5,cut);
cut1=*skymod-cut;
cut2=*skymod+cut;
redo=0;
istep=sign1(cut1-*(sky+minimm+1));

```

```

jstep=(istep+1)/2;
if(istep>0)
    goto dudz;
duzz: ;
    if((istep<0)&&(minimm<=0))
        goto duzz;
    if((*sky+minimm)<=cut1)&&((*sky+minimm+1)>=cut1))
        goto duzz;
dudz: ;
    delta=*(sky+minimm+jstep)-skymid;
    sum=sum-(double)istep*(double)delta;
    sumsq=sumsq-(double)istep*(double)(delta*delta);
    minimm=minimm+istep;
    redo=1;
    goto duzz;
duzz: ;
    istep=sign1(cut2-*(sky+maximm));
    jstep=(istep+1)/2;
    if(istep<0)
        goto dddz;
ddzz: ;
    if((istep>0)&&(maximm>=nsky))
        goto ddcz;
    if((*sky+maximm)<=cut2)&&((*sky+maximm+1)>=cut2))
        goto ddcz;
dddz: ;
    delta=*(sky+maximm+jstep)-skymid;
    sum=sum+(double)istep*(double)delta;
    sumsq=sumsq+(double)istep*(double)(delta*delta);
    maximm=maximm+istep;
    redo=1;
    goto ddzz;
ddcz: ;
    *skymn=(float)((sum)/((double)(maximm-minimm)));
    *sigma=(float)(sqrt(sumsq/(double)(maximm-minimm)-
        (double)((*skymn)*(*skymn))));
    *skymn=*skymn+skymid;
    *skymed=0.0;
    x=0.0;
    center=(float)(minimm+1+maximm)/2.;
    side=(float)((int)(0.05*(float)(maximm-minimm)))/2.+0.25;
    a=(int)(center-side);
    b=(int)(center+side);
    for(i=a;i<=b;i++)
    {
        *skymed=*(sky+i)+*skymed;
        x=x+1;
    }
    *skymed=(*skymed)/x;
    *skymod=*skymn;

```

```

if (*skymed < *skymn)
    *skymod = 3. * (*skymed) - 2. * (*skymn);
if (redo == 1)
    goto dzzz;
*skew = (*skymn - *skymod) / amax1(1., *sigma);
nsky = maximm - minimm;
return(0);

nnzz: ;
    *sigma = -1.0;
    *skew = 0.0;
    return(0);
}

////////////////////////////////////
// LOAD
////////////////////////////////////

int load(char *nomefile, int colpar, int flag)
{
    unsigned int i, j, k,
                a, b, c[3],
                col1, col2;

    FILE *fp;

    fp = fopen(nomefile, "r+b");
    if (fp == NULL)
        return(1);

    for (k = 0; k < nrow; k++)
    {
        for (j = 0; j < (ncol/2); j++)
        {
            for (i = 0; i < 3; i++) {
                c[i] =getc(fp);}
            a = ((c[1] << 12) >> 4) | c[0];
            b = ((c[1] >> 4) << 8) | c[2];
            if (flag == 1)
            {
                a = *(hhh + ncol*k + 2*j) - a;
                b = *(hhh + ncol*k + 2*j + 1) - b;
            }
            //a = *(hhh + 192*k + 2*j) - a;
            //b = *(hhh + 192*k + 2*j + 1) - b;
            *(hhh + ncol*k + 2*j) = a;
            *(hhh + ncol*k + 2*j + 1) = b;
        }
        //*(hhh + 192*k + 2*j) = a;
        //*(hhh + 192*k + 2*j + 1) = b;
        col1 = a / colpar;
        col2 = b / colpar;
    }
}

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 159/212
File: AmicaSoftware.doc

```
        //putpixel(300+2*j,50+k,col1); // to output device only
        //putpixel(301+2*j,50+k,col2); // to output device only
    }
}
fclose(fp);
return(0);
}

////////////////////////////////////
// START
////////////////////////////////////

void start(void)
{
    char  stringa[11][30];
    float vax;
    int   i;

    opt[0] = 1.6;
    strcpy(stringa[0],"READ NOISE (ADU)");
    opt[1] = 54.;
    strcpy(stringa[1],"GAIN (e-/ADU)");
    opt[2] = 8.;
    strcpy(stringa[2],"LOW GOOD DATUM (in sigmas)");
    opt[3] = 3300.;
    strcpy(stringa[3],"HIGH GOOD DATUM (in ADU)");
    opt[4] = 2.5;
    strcpy(stringa[4],"FWHM OF OBJECT");
    opt[5] = 40.;
    strcpy(stringa[5],"THRESHOLD (in sigmas)");
    opt[6] = 0.0;
    strcpy(stringa[6],"LOW SHARPNESS CUTOFF");
    opt[7] = 1.1;
    strcpy(stringa[7],"HIGH SHARPNESS CUTOFF");
    opt[8] = -2.0;
    strcpy(stringa[8],"LOW ROUNDNESS CUTOFF");
    opt[9] = 2.0;
    strcpy(stringa[9],"HIGH ROUNDNESS CUTOFF");
}

////////////////////////////////////
// ROUT1
////////////////////////////////////

int rout1(const float *alfa, const float *beta)
{
    int sug;

    if(*alfa<*beta)
        sug=-1;
}
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 160/212
File: AmicaSoftware.doc

```
    if(*alfa==*beta)
        sug=0;
    if(*alfa>*beta)
        sug=1;
    return(sug);
}

/////////////////////////////////////////////////////////////////
// AMIN1
/////////////////////////////////////////////////////////////////

float amin1(float x,float y)
{
    float minus;
    if(x<y)
        minus=x;
    else
        minus=y;
    return(minus);
}

/////////////////////////////////////////////////////////////////
// AMAX1
/////////////////////////////////////////////////////////////////

float amax1(float x,float y)
{
    float max;
    if(x<y)
        max=y;
    else
        max=x;
    return(max);
}

/////////////////////////////////////////////////////////////////
// SIGN1
/////////////////////////////////////////////////////////////////

int sign1(float x)
{
    int segno;

    if(fabs(x)>x)
        segno=-1;
    else
        segno=1;
    return(segno);
}
```

```
////////////////////////////////////  
// CENTRO  
////////////////////////////////////  
  
void centro(int x, int y, int h)  
{  
    int a,i,j,conta;  
    float rox[ncol],roy[nrow],xx,yy,sum,som,xuno,yuno;  
  
    a=4;  
    conta=0;  
giro: ;  
    conta=conta+1;  
  
    for(i=-a;i<=a;i++)  
    {  
        rox[x+i]=0.;  
        for(j=-a;j<=a;j++)  
        {  
            rox[x+i]=rox[x+i]+(float)*(hhh+(y+j)*ncol+x+i);  
        }  
    }  
  
    for(j=-a;j<=a;j++)  
    {  
        roy[y+j]=0.;  
        for(i=-a;i<=a;i++)  
        {  
            roy[y+j]=roy[y+j]+(float)*(hhh+(y+j)*ncol+x+i);  
        }  
    }  
  
    xx=0.;  
    for(i=-a;i<=a;i++)  
    {  
        xx=xx+rox[x+i];  
    }  
    xx=xx/((float)(2*a+1));  
  
    yy=0.;  
    for(j=-a;j<=a;j++)  
    {  
        yy=yy+roy[y+j];  
    }  
    yy=yy/((float)(2*a+1));  
  
    sum=0.;  
    som=0.;  
    for(i=-a;i<=a;i++)  
    {
```

```

        if(rox[x+i]>=xx)
        {
            sum=sum+(rox[x+i]-xx)*(float)(x+i);
            som=som+(rox[x+i]-xx);
        }
    }
    xuno=sum/som;

    sum=0.;
    som=0.;
    for(j=-a;j<=a;j++)
    {
        if(roy[y+j]>=yy)
        {
            sum=sum+(roy[y+j]-yy)*(float)(y+j);
            som=som+(roy[y+j]-yy);
        }
    }
    yuno=sum/som;

    xx=xuno-(float)x;
    xx=xx*xx;
    yy=yuno-(float)y;
    yy=yy*yy;
    if((yy<1.0)&&(xx<1.0))
    {
        centro_x[h]=xuno;
        centro_y[h]=yuno;
        return;
    }
    x=(int)(xuno+0.5);
    y=(int)(yuno+0.5);
    //if((conta>100)|(x<a)|(x>191-a)|(y<a)|(y>164-a))
    if((conta>100)|(x<a)|(x>ncol-1-a)|(y<a)|(y>nrow-1-a))
    {
        centro_x[h]=1000.;
        centro_y[h]=1000.;
        return;
    }

    goto giro;
}

////////////////////////////////////
// CheckFOV
////////////////////////////////////

bool CheckFOV(void)
{
    bool bClose;

```

```
bClose = false;

// check (FOV center versus CurRegion in Star Catalogue)

// ...

bCloseToBorder = bClose;
return bClose;
}

/////////////////////////////////////////////////////////////////
// SEEING
/////////////////////////////////////////////////////////////////

void seeing(float xcen, float ycen, float sky, float *FWHM, float *sigma)
{
    float  dum,dam,HalfWidth[4],HalfMax,sum;
    int    ix,jy,i,j,k,marg,num;

    ix=(int)(xcen+0.5);
    jy=(int)(ycen+0.5);

    HalfMax=((float)*(hhh+jy*ncol+ix))-sky)/2.;

    marg=min(8,ncol-ix);
    if(marg>1)
    {
        k=0;
        do
        {
            k=k+1;
            dum=(float)*(hhh+jy*ncol+ix+k-1))-sky;
            dam=(float)*(hhh+jy*ncol+ix+k))-sky;
        }
        while((dam>HalfMax)&&(k<marg));
        if((k<marg)&&(dum>dam))
            HalfWidth[0] = (float)(ix+k) - ((HalfMax-dam)/(dum-dam)) - xcen;
        else
            HalfWidth[0] = 0.0;
    }
    else
        HalfWidth[0] = 0.0;

    marg=min(8,ix);
    if(marg>1)
    {
        k=0;
        do
        {
```

```
k=k-1;
dum=(float)*(hhh+jy*ncol+ix+k+1))-sky;
dam=(float)*(hhh+jy*ncol+ix+k))-sky;
}
while((dam>HalfMax)&&(k>-marg));
if((k>-marg)&&(dum>dam))
  HalfWidth[1] = xcen - (float)(ix+k) - ((HalfMax-dam)/(dum-dam));
else
  HalfWidth[1] = 0.0;
}
else
  HalfWidth[1] = 0.0;

marg=min(8,ncol-jy);
if(marg>1)
{
  k=0;
  do
  {
    k=k+1;
    dum=(float)*(hhh+(jy+k-1)*ncol+ix))-sky;
    dam=(float)*(hhh+(jy+k)*ncol+ix))-sky;
  }
  while((dam>HalfMax)&&(k<marg));
  if((k<marg)&&(dum>dam))
  {
    HalfWidth[2] = (float)(jy+k) - ((HalfMax-dam)/(dum-dam)) - ycen;
    HalfWidth[2] = VC*HalfWidth[2];
  }
  else
    HalfWidth[2] = 0.0;
}
else
  HalfWidth[2] = 0.0;

marg=min(8,jy);
if(marg>1)
{
  k=0;
  do
  {
    k=k-1;
    dum=(float)*(hhh+(jy+k+1)*ncol+ix))-sky;
    dam=(float)*(hhh+(jy+k)*ncol+ix))-sky;
  }
  while((dam>HalfMax)&&(k>-marg));
  if((k>-marg)&&(dum>dam))
  {
```

```
        HalfWidth[3] = ycen - (float)(jy+k) - ((HalfMax-dam)/(dum-dam));
        HalfWidth[3] = VC*HalfWidth[3];
    }
    else
        HalfWidth[3] = 0.0;
}
else
    HalfWidth[3] = 0.0;

sum=0.0;
num=0;
for(k=0;k<4;k++)
{
    if(HalfWidth[k]>0)
    {
        sum = sum+HalfWidth[k];
        num = num+1;
    }
}

if(num>1)
{
    *FWHM = sum/(float)num;

    sum=0.0;
    num=0;
    for(k=0;k<4;k++)
    {
        if(HalfWidth[k]>0)
        {
            sum = sum+(*FWHM-HalfWidth[k])*(*FWHM-HalfWidth[k]);
            num = num+1;
        }
    }
    *sigma = 2.0*(float)sqrt((double)sum)/((float)(num-1));
    *FWHM = 2.0*(*FWHM);
}
else
{
    *FWHM = 99.99;
    *sigma = 9.99;
}
return;
}
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 166/212
File: AmicaSoftware.doc

5.3.12. Tracker

```
//////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: tracker.c  
//  
// Version: 1.0  
// Modified: 18/06/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: Main application  
//  
//////////////////////////////////////  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
#include "def21020.h" /* Bit definitions for the registers */  
#include "defs.h" /* Loader values and addresses definitions */  
#include <21020.h> /* Needed for set_flag */  
#include <signal.h> /* Needed for interrupt(), raise(), signal() */  
  
#define TRUE 1  
#define FALSE 0  
#define PAGE_DIM 0x8000  
#define BUF_LEN 512  
#define MAX_CMD 20  
#define WBUFLen 0x100  
#define PRG_LEN 0x100  
#define LWAIT 2000000L  
#define HKSTDBY 100000L  
  
//////////////////////////////////////  
// global variables  
//////////////////////////////////////  
  
int timer_expired;  
int serial_int;  
int hkdata_int;  
int hk_val[16];  
int bmsg1std;  
int bmsg2std;  
int iNBytes1;  
int iNBytes2;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 167/212
File: AmicaSoftware.doc

```
unsigned char buf1[BUF_LEN];
unsigned char buf2[BUF_LEN];
int iBuf1In;
int iBuf2In;
unsigned char bufPrg[PRG_LEN];
int iBufPrIn;
unsigned char cmdque[MAX_CMD][BUF_LEN];
int iCmdIndx;
int bQueFull;

/* CRC16 utilities */
unsigned long crc_tab[256];
int computed=0;

int iwait = 0; /* defined here as a global; used in ASM file as _iwait */
long so_addr=0;
long de_addr=0;
extern int asm_var; /* defined in ASM file as _asm_var */
extern int asm_lp;
extern int asm_wre;
extern int asm_dle;

////////////////////////////////////
// function prototypes
////////////////////////////////////

void timer_handler(int signal);
void serialrx_handler(int signal);
void housekeeping_handler(int signal);
void timer_ops(void);
void serialrx_ops(void);
void housekeeping_ops(void);
void delay_short(void);
void delay_long(void);
void wait(int idelay);
unsigned char GetSerChar(void);
int SendSerChar(unsigned char ch, int port);
int ReadSerStatusReg(int serial);
unsigned char ReadSerDataReg(int serial);
void WriteSerDataReg(unsigned char ch, int serial);
void make_crc_table(void);
unsigned long update_crc(unsigned long crc, volatile unsigned char *buf, int len);
unsigned long crc(volatile unsigned char *buf, int len);
int CheckMessageCrc(int iPort);

/* assembly functions prototyped here */
void asm_wait(void);

////////////////////////////////////
// void main()
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 168/212
File: AmicaSoftware.doc

////////////////////////////////////

```
void main(void)
{
    int bFirstTime=1;
    int bRawPtDataAvailable;
    int bOkFullSearch;
    int bOkLocalSearch;
    int bOkTrack;
    int bFullSearchFailed=0;
    int bPrgExit=0;
    int iFullErrCount,iFullErrMaxCount;
    int iLocalErrCount,iLocalErrMaxCount;
    int iTrackErrCount,iTrackErrMaxCount;

    timer_expired=0;
    serial_int=0;
    hkdata_int=0;
    iFullErrMaxCount=1; // 1 = exit on first error
    iLocalErrMaxCount=1; // 1 = exit on first error
    iTrackErrMaxCount=1; // 1 = exit on first error

    iCmdIndx=0;

    /* CLR_FLAG,SET_FLAG */
    set_flag(SET_FLAG0, CLR_FLAG); /* camera 0 power off */
    set_flag(SET_FLAG1, CLR_FLAG); /* camera 1 power off */
    /*set_flag(SET_FLAG2, CLR_FLAG); /* choose image ram 0 */

    /* disable interrupts */
    interrupt(SIG_TMZ0,SIG_IGN); /* disable Timer = 0 (high priority option) */
    interrupt(SIG_IRQ3,SIG_IGN);/* disable Interrupt 3 - serial port read */
    interrupt(SIG_IRQ2,SIG_IGN);/* disable Interrupt 2 - image ready */
    interrupt(SIG_IRQ1,SIG_IGN);/* disable Interrupt 1 - serial port write */
    interrupt(SIG_IRQ0,SIG_IGN);/* disable Interrupt 0 - ADC data ready */
    interrupt(SIG_TMZ,SIG_IGN);/* disable Timer = 0 (low priority option) */
    interrupt(SIG_USR0,SIG_IGN); /* disable User software interrupt 0 */
    interrupt(SIG_USR1,SIG_IGN); /* disable User software interrupt 1 */
    interrupt(SIG_USR2,SIG_IGN); /* disable User software interrupt 2 */
    interrupt(SIG_USR3,SIG_IGN); /* disable User software interrupt 3 */
    interrupt(SIG_USR4,SIG_IGN); /* disable User software interrupt 4 */
    interrupt(SIG_USR5,SIG_IGN); /* disable User software interrupt 5 */

    /* clear pending interrupts */
    clear_interrupt(SIG_TMZ0); /* clear Timer = 0 (high priority option) */
    clear_interrupt(SIG_IRQ3); /* clear Interrupt 3 - serial port read */
    clear_interrupt(SIG_IRQ2); /* clear Interrupt 2 - image ready */
    clear_interrupt(SIG_IRQ1); /* clear Interrupt 1 - serial port write */
    clear_interrupt(SIG_IRQ0); /* clear Interrupt 0 - ADC data ready */
    clear_interrupt(SIG_TMZ); /* clear Timer = 0 (low priority option) */
}
```

```

clear_interrupt(SIG_USR0);          /* clear User software interrupt 0 */
clear_interrupt(SIG_USR1);          /* clear User software interrupt 1 */
clear_interrupt(SIG_USR2);          /* clear User software interrupt 2 */
clear_interrupt(SIG_USR3);          /* clear User software interrupt 3 */
clear_interrupt(SIG_USR4);          /* clear User software interrupt 4 */
clear_interrupt(SIG_USR5);          /* clear User software interrupt 5 */

/*      set 10s timer and enable Timer interrupt */
/* 20Mhz -> clock cycle 50ns -> 1s=20.000.000 cycles */
interrupt(SIG_TMZ, timer_handler); /* set timer interrupt routine */
timer_set((unsigned int)200000000,(unsigned int)200000000); /*      period, current count
*/
timer_on();                          /* start timer */

/* enable Serial Port interrupt */
interrupt(SIG_IRQ3, serialrx_handler);

/*      enable Housekeeping data ready interrupt */
interrupt(SIG_IRQ0, housekeeping_handler);

//int iBootCondt;
//iBootCondt = GetBootCondition();
// special operations based on iBootCondt value

iFullErrCount=0;

/* loop and respond to interrupts */
while(1)
{
    idle(); /* return from this function after an interrupt */

    if(timer_expired) /* if the timer has expired, clear flag and service it */
    {
        timer_expired=0;
        timer_ops();
    }

    if(serial_int) /* if irq3 has occurred, clear flag and service it */
    {
        serial_int=0;
        serialrx_ops();
    }

    if(hkdata_int) /* if irq0 has occurred, clear flag and service it */
    {
        hkdata_int=0;
        housekeeping_ops();
    }

    if(iCmdIndx>0) /* if present, manage the oldest command in the queue */

```

```
{
    /* servicing one command in one main loop is enough */
    ManageCmdQueue();
}

if(iImgReady)
{
    if(bFirstTime)
    {
        //bRawPtDataAvailable = CheckForRawPtData();
        bFirstTime=0;
    }
    else
    {
        bRawPtDataAvailable=0;
    }

    if(!bRawPtDataAvailable)
    {
        bOkFullSearch=FullSearch();
    }
    else
    {
        // raw pointing data now available

        bOkFullSearch=1;
        bRawPtDataAvailable=1;
    }

    if(bOkFullSearch)
    {
        bOkLocalSearch=1;
        iFullErrCount=0;
        iLocalErrCount=0;

        while(bOkLocalSearch)
        {
            // (fast) search around known coordinates

            bOkLocalSearch = LocalSearch();

            if(bOkLocalSearch)
            {
                InitTrackFunction();
                bOkTrack=1;
                iLocalErrCount=0;
                iTrackErrCount=0;

                while(bOkTrack)
                {
```

```
        bOkTrack = TrackStars();

        if(!bOkTrack)
        {
            iTrackErrCount++;

            if(iTrackErrCount<iTrackErrMaxCount)
            {
                bOkTrack=1;
            }
        }
        // tracking lost (for "iTrackErrMaxCount" consecutive times)
    }
    else
    {
        // local search failed
        // try to repeat local search (for a limited number of times)
        iLocalErrCount++;
        if(iLocalErrCount<iLocalErrMaxCount)
        {
            // retry
            bOkLocalSearch=1;
        }
        // with more than 'iLocalErrMaxCount' errors returns to
FullSearch
    }
}
// local search failed
}
else
{
    // full search failed; retry 'iFullErrMaxCount' times, or reset
    iFullErrCount++;
    if(iFullErrCount<iFullErrMaxCount)
    {
        // retry
        bOkFullSearch=1;
    }
    else
    {
        // with more than 'iFullErrMaxCount' errors reset
        bPrgExit=1;
        bFullSearchFailed=1;
    }
}
}
}
}
```

```
void timer_handler(int signal)
{
    timer_expired=1;
}

void serialrx_handler(int signal)
{
    serial_int=1;
}

void housekeeping_handler(int signal)
{
    hkdata_int=1;
}

void timer_ops(void)
{
    /*    TIMER interrupt handler */

    timer_off();
    interrupt(SIG_TMZ, SIG_IGN);          /* disable timer interrupt */
    clear_interrupt(SIG_TMZ);            /* Timer = 0 (low priority option) */

    interrupt(SIG_IRQ3, SIG_IGN);
    clear_interrupt(SIG_IRQ3);

    /* enable serial interrupt service and wait for commands */

    clear_interrupt(SIG_IRQ3);
    interrupt(SIG_IRQ3, serialrx_handler);
}

void serialrx_ops(void)
{
    /*    SERIAL port received data interrupt handler */

    /*
    interrupt(SIG_IRQ3, SIG_IGN);
    clear_interrupt(SIG_IRQ3);
    */

    // ser0_data    0x80000000 // serial 0 data DM address
    // ser0_reg     0x80000005 // serial 0 register DM address
    // ser1_data    0x80000010 // serial 1 data DM address
    // ser1_reg     0x80000015 // serial 1 register DM address

    int msg1comp;
    int msg2comp;
    unsigned char cNewChar;
```

```
msg1comp=FALSE;
msg2comp=FALSE;

// store received chars in buffer

cNewChar=GetSerChar();

if(cNewChar<0x100) // valid
{
    if(iSer1)
        buf1[iBuf1In]=cNewChar;
    else
        buf2[iBuf2In]=cNewChar;
}
else
{
    return;
}

if(iSer1)
{
    if(!bmsg1std)
    {
        // initial condition; look for sync byte

        if(cNewChar==0xE5)
        {
            // sync byte: candidate start of new message
            bmsg1std=TRUE;
            iBuf1In++;
        }
    }
    else
    {
        // message bytes

        if(iBuf1In==1)
        {
            // message nr_bytes
            iNBytes1=cNewChar;
        }

        iBuf1In++;

        if(iBuf1In==iNBytes1)
        {
            // message completed
            // prepare for the next message
            bmsg1std=FALSE;
        }
    }
}
```

```
        if(CheckMessageCrc(1))
        {
            // message consistency checked
            // enable message interpreter
            msg1comp=TRUE;
        }
    }

    if(msg1comp) // add message to command queue
    {
        if(iCmdIndx<MAX_CMD-1)
        {
            int i;
            for(i=0;i<iNBytes1;i++)
            {
                cmdque[iCmdIndx][i]=buf1[i];
            }
            iCmdIndx++; // increase queue index
        }
        else // queue full
        {
            bQueFull=TRUE;
        }
    }
}
else
{
    if(!bmsg2std)
    {
        // initial condition; look for sync byte

        if(cNewChar==0xE5)
        {
            // sync byte: candidate start of new message
            bmsg2std=TRUE;
            iBuf2In++;
        }
    }
    else
    {
        // message bytes

        if(iBuf2In==1)
        {
            // message nr_bytes
            iNBytes2=cNewChar;
        }

        iBuf2In++;
    }
}
```

```
        if(iBuf2In==iNBytes2)
        {
            // message completed
            // prepare for the next message
            bmsg2std=FALSE;

            if(CheckMessageCrc(2))
            {
                // message consistency checked
                // enable message interpreter
                msg2comp=TRUE;
            }
        }
    }

    if(msg2comp) // add message to command queue
    {
        if(iCmdIndx<MAX_CMD-1)
        {
            int i;
            for(i=0;i<iNBytes2;i++)
            {
                cmdque[iCmdIndx][i]=buf2[i];
            }
            iCmdIndx++; // increase queue index
        }
        else // queue full
        {
            bQueFull=TRUE;
        }
    }
}
// exit; commands queue manager is at the end of the main loop
}

void housekeeping_ops(void)
{
    /*    FPGA housekeeping data ready interrupt handler */

    int value;

    /*
    interrupt(SIG_IRQ0, SIG_IGN);
    clear_interrupt(SIG_IRQ0);
    */

    /* read ADC values and store to global variables */
}
```

```

asm( "%0=dm(0x80000030);"."=d" (value) ); // adc1_ch0
hk_val[0]=value;
asm( "%0=dm(0x80000031);"."=d" (value) ); // adc1_ch1
hk_val[1]=value;
asm( "%0=dm(0x80000032);"."=d" (value) ); // adc1_ch2
hk_val[2]=value;
asm( "%0=dm(0x80000033);"."=d" (value) ); // adc1_ch3
hk_val[3]=value;
asm( "%0=dm(0x80000034);"."=d" (value) ); // adc1_ch4
hk_val[4]=value;
asm( "%0=dm(0x80000035);"."=d" (value) ); // adc1_ch5
hk_val[5]=value;
asm( "%0=dm(0x80000036);"."=d" (value) ); // adc1_ch6
hk_val[6]=value;
asm( "%0=dm(0x80000037);"."=d" (value) ); // adc1_ch7
hk_val[7]=value;
asm( "%0=dm(0x80000038);"."=d" (value) ); // adc2_ch0
hk_val[8]=value;
asm( "%0=dm(0x80000039);"."=d" (value) ); // adc2_ch1
hk_val[9]=value;
asm( "%0=dm(0x8000003A);"."=d" (value) ); // adc2_ch2
hk_val[10]=value;
asm( "%0=dm(0x8000003B);"."=d" (value) ); // adc2_ch3
hk_val[11]=value;
asm( "%0=dm(0x8000003C);"."=d" (value) ); // adc2_ch4
hk_val[12]=value;
asm( "%0=dm(0x8000003D);"."=d" (value) ); // adc2_ch5
hk_val[13]=value;
asm( "%0=dm(0x8000003E);"."=d" (value) ); // adc2_ch6
hk_val[14]=value;
asm( "%0=dm(0x8000003F);"."=d" (value) ); // adc2_ch7
hk_val[15]=value;
}

void delay_short()
{
    // ser_rit = 128 delay cycles for ram, 32 for flash
    wait(128);
}

void delay_long()
{
    // ser_rit*2 = 256 delay cycles for ram, 64 for flash
    wait(256);
}

void wait(int idelay)
{
    iwait=idelay;
    asm_wait();
}

```

```
}  
  
unsigned char GetSerChar(void)  
{  
    int value;  
    unsigned char ch;  
  
    // try serial 0  
  
    value=ReadSerStatusReg(0);  
  
    // if bit0 = 0 (RxRDY; receive data ready) change serial  
  
    if((int)(value/2)*2==value) // RxRDY=0 -> read serial 1  
    {  
        value=ReadSerStatusReg(1);  
  
        if((int)(value/2)*2==value) // RxRDY=0; no data to read  
        {  
            return 0xffff;  
        }  
        else // RxRDY=1; read received data  
        {  
            iSer1=0;  
            delay_short();  
            ch = ReadSerDataReg(1); // read data register ser1_data  
            return ch;  
        }  
    }  
    else // RxRDY=1; read received data  
    {  
        iSer1=1;  
        delay_short();  
        ch = ReadSerDataReg(0); // read data register ser0_data  
        return ch;  
    }  
  
    return ch;  
}  
  
int SendSerChar(unsigned char ch, int port)  
{  
    // port can be 0 or 1  
    // returns 1 if success; if 0 the caller is responsible of trying again  
  
    int value;  
    delay_long();  
  
    if(port==0)  
    {
```

```
        value=ReadSerStatusReg(0);

        // if bit5 = 0 try again

        value=(int)(value/32); // :32 -> rshift5

        if((int)(value/2)*2==value)
        {
            return 0;           // THRE=0; transmitter holding register empty
        }
        else
        {
            WriteSerDataReg(ch,0); // write data register ser0_data
            delay_short();
            return 1;           // done
        }
    }
else if(port==1)
{
    value=ReadSerStatusReg(1);

    // if bit5 = 0 try again

    value=(int)(value/32); // :32 -> rshift5

    if((int)(value/2)*2==value)
    {
        return 0;           // LSB=0
    }
    else
    {
        WriteSerDataReg(ch,1); // write data register ser1_data
        delay_short();
        return 1;           // done
    }
}

return 0; // just in case of problems
}

int ReadSerStatusReg(int serial)
{
    // read status register ser0_reg or ser1_reg

    int value=0;

    if(serial==0)
    {
        asm( "%0=dm(0x80000005);":"=d" (value) ); // ser0_reg
    }
}
```

```
    else if(serial==1)
    {
        asm( "%0=dm(0x80000015);":"=d" (value) ); // ser1_reg
    }
    return value;
}

unsigned char ReadSerDataReg(int serial)
{
    // read data register ser0_data or ser1_data

    unsigned char ch;

    if(serial==0)
    {
        asm( "%0=dm(0x80000000);":"=d" (ch) ); // ser0_data
    }
    else if(serial==1)
    {
        asm( "%0=dm(0x80000010);":"=d" (ch) ); // ser1_data
    }
    return ch;
}

void WriteSerDataReg(unsigned char ch, int serial)
{
    // write data register ser0_data or ser1_data

    if(serial==0)
    {
        asm( "dm(0x80000000)=%0;":"=d" (ch) ); // ser0_data
    }
    else if(serial==1)
    {
        asm( "dm(0x80000010)=%0;":"=d" (ch) ); // ser1_data
    }
}

int CheckMessageCrc(int iPort)
{
    unsigned char ucCrcLo;
    unsigned char ucCrcHi;
    unsigned long CrcRead;
    unsigned long CrcCalc;
    int iResult;

    if(iPort==1)
    {
        ucCrcLo=buf1[iBuf1In-1];
        ucCrcHi=buf1[iBuf1In-2];
    }
}
```

```
        CrcRead=ucCrcLo+(ucCrcHi*256);    // x256->lshiftx8

        // unsigned long crc(unsigned char *buf, int len);
        CrcCalc = crc(buf1,iBuf1In-1);
    }
    else
    {
        ucCrcLo=buf2[iBuf2In-1];
        ucCrcHi=buf2[iBuf2In-2];
        CrcRead=ucCrcLo+(ucCrcHi*256);    // x256->lshiftx8

        // unsigned long crc(unsigned char *buf, int len);
        CrcCalc = crc(buf2,iBuf2In-1);
    }

    if(CrcRead==CrcCalc)
        iResult=1;
    else
        iResult=0;

    return iResult;
}

void make_crc_table(void)
{
    // Build a table for a fast CRC calculation

    unsigned long c;
    int n, k;

    for(n = 0; n < 256; n++)
    {
        c = (unsigned long) n;
        for(k = 0; k < 8; k++)
        {
            if(c & 1)
                c = 0xedb88320L ^ (c >> 1);
            else
                c = c >> 1;
        }
        crc_tab[n] = c;
    }
    computed = 1;
}

unsigned long update_crc(unsigned long crc, volatile unsigned char *buf, int len)
{
    // Update a running CRC with the bytes buf[0..len-1].
    // The CRC should be initialized to all 1's, and the transmitted
    // value is the 1's complement of the final running CRC.
```

```
unsigned long c = crc;
int n;

if(!computed)
    make_crc_table();
for(n = 0; n < len; n++)
{
    c = crc_tab[(c ^ buf[n]) & 0xff] ^ (c >> 8);
}
return c;
}

unsigned long crc(volatile unsigned char *buf, int len)
{
    // Return the CRC of the bytes buf[0..len-1].

    return update_crc(0xffffffffL, buf, len) ^ 0xffffffffL;
}
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 182/212
File: AmicaSoftware.doc

5.3.13. tracker1.c

```
////////////////////////////////////
//
// AMICA FOR AMS
//
// File: tracker.c
//
// Version: 1.0
// Modified: 04/06/07
// Author: F.Roncella
// Hardware: ADSP-21020
// Project: StarTracker
//
// Notes: Main application
//
////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "def21020.h" /* Bit definitions for the registers */
#include "defs.h" /* Loader values and addresses definitions */
#include <21020.h> /* Needed for set_flag */
#include <signal.h> /* Needed for interrupt(), raise(), signal() */

#define TRUE 1
#define FALSE 0

////////////////////////////////////
// global variables
////////////////////////////////////

int timer_expired;
int serial_int;
int hkdata_int;
int hk_val[16];
int iCmdIdx;
int iwait = 0; /* defined here as a global; used in ASM file as _iwait */
extern int asm_lp; /* defined in ASM file as _asm_lp */

////////////////////////////////////
// function prototypes
////////////////////////////////////

void timer_handler(int signal);
void serialrx_handler(int signal);
void housekeeping_handler(int signal);
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 183/212
File: AmicaSoftware.doc

```
void timer_ops(void);
void serialrx_ops(void);
void housekeeping_ops(void);
void delay_short(void);
void delay_long(void);
void wait(int idelay);

/* assembly functions prototyped here */
void asm_wait(void);

////////////////////////////////////////////////////////////////
// void main()
////////////////////////////////////////////////////////////////

void main(void)
{
    timer_expired=0;
    serial_int=0;
    hkdata_int=0;

    iCmdIndx=0;

    /* CLR_FLAG,SET_FLAG */
    set_flag(SET_FLAG0, CLR_FLAG); /* camera 0 power off */
    set_flag(SET_FLAG1, CLR_FLAG); /* camera 1 power off */
    /*set_flag(SET_FLAG2, CLR_FLAG); /* choose image ram 0 */

    /* disable interrupts */
    interrupt(SIG_TMZ0,SIG_IGN); /* disable Timer = 0 (high priority option) */
    interrupt(SIG_IRQ3,SIG_IGN);/* disable Interrupt 3 - serial port read */
    interrupt(SIG_IRQ2,SIG_IGN);/* disable Interrupt 2 - image ready */
    interrupt(SIG_IRQ1,SIG_IGN);/* disable Interrupt 1 - serial port write */
    interrupt(SIG_IRQ0,SIG_IGN);/* disable Interrupt 0 - ADC data ready */
    interrupt(SIG_TMZ,SIG_IGN);/* disable Timer = 0 (low priority option) */
    interrupt(SIG_USR0,SIG_IGN); /* disable User software interrupt 0 */
    interrupt(SIG_USR1,SIG_IGN); /* disable User software interrupt 1 */
    interrupt(SIG_USR2,SIG_IGN); /* disable User software interrupt 2 */
    interrupt(SIG_USR3,SIG_IGN); /* disable User software interrupt 3 */
    interrupt(SIG_USR4,SIG_IGN); /* disable User software interrupt 4 */
    interrupt(SIG_USR5,SIG_IGN); /* disable User software interrupt 5 */

    /* clear pending interrupts */
    clear_interrupt(SIG_TMZ0); /* clear Timer = 0 (high priority option) */
    clear_interrupt(SIG_IRQ3); /* clear Interrupt 3 - serial port read */
    clear_interrupt(SIG_IRQ2); /* clear Interrupt 2 - image ready */
    clear_interrupt(SIG_IRQ1); /* clear Interrupt 1 - serial port write */
    clear_interrupt(SIG_IRQ0); /* clear Interrupt 0 - ADC data ready */
    clear_interrupt(SIG_TMZ); /* clear Timer = 0 (low priority option) */
    clear_interrupt(SIG_USR0); /* clear User software interrupt 0 */
    clear_interrupt(SIG_USR1); /* clear User software interrupt 1 */
}
```

```

clear_interrupt(SIG_USR2);          /* clear User software interrupt 2 */
clear_interrupt(SIG_USR3);          /* clear User software interrupt 3 */
clear_interrupt(SIG_USR4);          /* clear User software interrupt 4 */
clear_interrupt(SIG_USR5);          /* clear User software interrupt 5 */

/*      set 10s timer and enable Timer interrupt */
/* 20Mhz -> clock cycle 50ns -> 1s=20.000.000 cycles */
interrupt(SIG_TMZ, timer_handler); /* set timer interrupt routine */
timer_set((unsigned int)200000000,(unsigned int)200000000); /*      period, current count
*/
timer_on();                          /* start timer */

/* enable Serial Port interrupt */
interrupt(SIG_IRQ3, serialrx_handler);

/*      enable Housekeeping data ready interrupt */
interrupt(SIG_IRQ0, housekeeping_handler);

/* loop and respond to interrupts */
while(1)
{
    idle(); /* return from this function after an interrupt */

    if(timer_expired) /* if the timer has expired, clear flag and service it */
    {
        timer_expired=0;
        timer_ops();
    }

    if(serial_int) /* if irq3 has occurred, clear flag and service it */
    {
        serial_int=0;
        serialrx_ops();
    }

    if(hkdata_int) /* if irq0 has occurred, clear flag and service it */
    {
        hkdata_int=0;
        housekeeping_ops();
    }

    if(iCmdIndx>0) /* if present, manage the oldest command in the queue */
    {
        /* servicing one command in one main loop is enough */
        ManageCmdQueue();
    }
}

void timer_handler(int signal)

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 185/212
File: AmicaSoftware.doc

```
{
    timer_expired=1;
}

void serialrx_handler(int signal)
{
    serial_int=1;
}

void housekeeping_handler(int signal)
{
    hkdata_int=1;
}

void timer_ops(void)
{
    /*    TIMER interrupt handler */

    timer_off();
    interrupt(SIG_TMZ, SIG_IGN);           /* disable timer interrupt */
    clear_interrupt(SIG_TMZ);             /* Timer = 0 (low priority option) */

    interrupt(SIG_IRQ3, SIG_IGN);
    clear_interrupt(SIG_IRQ3);

    /* enable serial interrupt service and wait for commands */

    clear_interrupt(SIG_IRQ3);
    interrupt(SIG_IRQ3, serialrx_handler);
}

void serialrx_ops(void)
{
    /*    SERIAL port received data interrupt handler */

    /*
    interrupt(SIG_IRQ3, SIG_IGN);
    clear_interrupt(SIG_IRQ3);
    */

    /*    ser0_data    0x80000000 // serial 0 data DM address */
    /*    ser0_reg     0x80000005 // serial 0 register DM address */
    /*    ser1_data    0x80000010 // serial 1 data DM address */
    /*    ser1_reg     0x80000015 // serial 1 register DM address */
}

void housekeeping_ops(void)
{
    /*    FPGA housekeeping data ready interrupt handler */
```

```
int value;

/*
interrupt(SIG_IRQ0, SIG_IGN);
clear_interrupt(SIG_IRQ0);
*/

/* read ADC values and store to global variables */

asm( "%0=dm(0x80000030);":"=d" (value) ); // adc1_ch0
hk_val[0]=value;
asm( "%0=dm(0x80000031);":"=d" (value) ); // adc1_ch1
hk_val[1]=value;
asm( "%0=dm(0x80000032);":"=d" (value) ); // adc1_ch2
hk_val[2]=value;
asm( "%0=dm(0x80000033);":"=d" (value) ); // adc1_ch3
hk_val[3]=value;
asm( "%0=dm(0x80000034);":"=d" (value) ); // adc1_ch4
hk_val[4]=value;
asm( "%0=dm(0x80000035);":"=d" (value) ); // adc1_ch5
hk_val[5]=value;
asm( "%0=dm(0x80000036);":"=d" (value) ); // adc1_ch6
hk_val[6]=value;
asm( "%0=dm(0x80000037);":"=d" (value) ); // adc1_ch7
hk_val[7]=value;
asm( "%0=dm(0x80000038);":"=d" (value) ); // adc2_ch0
hk_val[8]=value;
asm( "%0=dm(0x80000039);":"=d" (value) ); // adc2_ch1
hk_val[9]=value;
asm( "%0=dm(0x8000003A);":"=d" (value) ); // adc2_ch2
hk_val[10]=value;
asm( "%0=dm(0x8000003B);":"=d" (value) ); // adc2_ch3
hk_val[11]=value;
asm( "%0=dm(0x8000003C);":"=d" (value) ); // adc2_ch4
hk_val[12]=value;
asm( "%0=dm(0x8000003D);":"=d" (value) ); // adc2_ch5
hk_val[13]=value;
asm( "%0=dm(0x8000003E);":"=d" (value) ); // adc2_ch6
hk_val[14]=value;
asm( "%0=dm(0x8000003F);":"=d" (value) ); // adc2_ch7
hk_val[15]=value;
}

void delay_short()
{
    // ser_rit = 128 delay cycles for ram, 32 for flash
    wait(128);
}

void delay_long()
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 187/212
File: AmicaSoftware.doc

```
{  
    // ser_rit*2 = 256 delay cycles for ram, 64 for flash  
    wait(256);  
}  
  
void wait(int idelay)  
{  
    iwait=idelay;  
    asm_wait();  
}
```

5.3.14. utils.c

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File:  utils.c  
//  
// Version: 1.0  
// Modified: 02/03/07  
// Author:  F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes:  various utility functions  
//  
//  
////////////////////////////////////  
  
#include "utils.h"  
  
////////////////////////////////////  
// void test( void )  
////////////////////////////////////  
  
void test( void )  
{  
    iTestCount=1;  
}  
  
void SommaImmagini(double **frameA, double **frameB, double **frameSum,  
                   int rigframe, int colframe)  
{  
    int i,j;  
  
    for(i=1;i<=rigframe;i++)  
    {  
        for(j=1;j<=colframe;j++)  
        {  
            frameSum[i][j]=frameA[i][j]+frameB[i][j];  
        }  
    }  
}  
  
void DifferenzaImmagini(double **frameA, double **frameB, double **frameDiff,  
                       int rigframe, int colframe)  
{  
    int i,j;
```

```
    for(i=1;i<=rigframe;i++)
    {
        for(j=1;j<=colframe;j++)
        {
            frameDiff[i][j]=frameA[i][j]-frameB[i][j];
        }
    }
}

void ImaginePerScalare(double **frameOrig, double **frameNew, double fattmol,
                    int rigframe, int colframe)
{
    int i,j;

    for(i=1;i<=rigframe;i++)
    {
        for(j=1;j<=colframe;j++)
        {
            frameNew[i][j]=fattmol*frameOrig[i][j];
        }
    }
}

void ImaginePiuScalare(double **frameOrig, double **frameNew, double fattadd,
                    int rigframe, int colframe)
{
    int i,j;

    for(i=1;i<=rigframe;i++)
    {
        for(j=1;j<=colframe;j++)
        {
            frameNew[i][j]=frameOrig[i][j]+fattadd;
        }
    }
}
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 190/212
File: AmicaSoftware.doc

5.3.15. def21020.h

```
/*
 *
 * def21020.h
 * STATUS REGISTER BIT DEFINITIONS FOR ADSP-21020.
 *
 * (c) Copyright 2001-2004 Analog Devices, Inc. All rights reserved.
 * $Revision: 1.1 $
 */
This include file contains a list of "defines" to enable the programmer to
use symbolic names for all of the system register bits for the ADSP-21020.
*/
#ifndef __DEF21020_H_
#define __DEF21020_H_

/* MODE1 register */
#define BR0 0x00000002 /* Bit 1: Bit-reverse for I0 (uses DMS0- only) */
#define SRCU 0x00000004 /* Bit 2: Alt. register select for comp. units */
#define SRD1H 0x00000008 /* Bit 3: DAG1 alt. register select (7-4) */
#define SRD1L 0x00000010 /* Bit 4: DAG1 alt. register select (3-0) */
#define SRD2H 0x00000020 /* Bit 5: DAG2 alt. register select (15-12) */
#define SRD2L 0x00000040 /* Bit 6: DAG2 alt. register select (11-8) */
#define SRRFH 0x00000080 /* Bit 7: Register file alt. select for R(15-8) */
#define SRRFL 0x00000400 /* Bit 10: Register file alt. select for R(7-0) */
#define NESTM 0x00000800 /* Bit 11: Interrupt nesting enable */
#define IRPTEN 0x00001000 /* Bit 12: Global interrupt enable */
#define ALUSAT 0x00002000 /* Bit 13: Enable ALU fixed-pt. saturation */
#define TRUNC 0x00008000 /* Bit 15: 1=fltg-pt. truncation 0=Rnd to nearest */
#define RND32 0x00010000 /* Bit 16: 1=32-bit fltg-pt.rounding 0=40-bit rnd */

/* MODE2 register */
#define IRQ0E 0x00000001 /* Bit 0: IRQ0- 1=edge sens. 0=level sens. */
#define IRQ1E 0x00000002 /* Bit 1: IRQ1- 1=edge sens. 0=level sens. */
#define IRQ2E 0x00000004 /* Bit 2: IRQ2- 1=edge sens. 0=level sens. */
#define IRQ3E 0x00000008 /* Bit 3: IRQ3- 1=edge sens. 0=level sens. */
#define CADIS 0x00000010 /* Bit 4: Cache disable */
#define TIMEN 0x00000020 /* Bit 5: Timer enable */
#define FLG0O 0x00008000 /* Bit 15: FLAG0 1=output 0=input */
#define FLG1O 0x00010000 /* Bit 16: FLAG1 1=output 0=input */
#define FLG2O 0x00020000 /* Bit 17: FLAG2 1=output 0=input */
#define FLG3O 0x00040000 /* Bit 18: FLAG3 1=output 0=input */
#define CAFRZ 0x00080000 /* Bit 19: Cache freeze */

/* ASTAT register */
#define AZ 0x00000001 /* Bit 0: ALU result zero or fltg-pt. underflow */
```

```

#define AV    0x00000002 /* Bit 1: ALU overflow */
#define AN    0x00000004 /* Bit 2: ALU result negative */
#define AC    0x00000008 /* Bit 3: ALU fixed-pt. carry */
#define AS    0x00000010 /* Bit 4: ALU X input sign (ABS and MANT ops) */
#define AI    0x00000020 /* Bit 5: ALU fltg-pt. invalid operation */
#define MN    0x00000040 /* Bit 6: Multiplier result negative */
#define MV    0x00000080 /* Bit 7: Multiplier overflow */
#define MU    0x00000100 /* Bit 8: Multiplier fltg-pt. underflow */
#define MI    0x00000200 /* Bit 9: Multiplier fltg-pt. invalid operation */
#define AF    0x00000400 /* Bit 10: ALU fltg-pt. operation */
#define SV    0x00000800 /* Bit 11: Shifter overflow */
#define SZ    0x00001000 /* Bit 12: Shifter result zero */
#define SS    0x00002000 /* Bit 13: Shifter input sign */
#define BTF   0x00040000 /* Bit 18: Bit test flag for system registers */
#define FLG0  0x00080000 /* Bit 19: FLAG0 value */
#define FLG1  0x00100000 /* Bit 20: FLAG1 value */
#define FLG2  0x00200000 /* Bit 21: FLAG2 value */
#define FLG3  0x00400000 /* Bit 22: FLAG3 value */
#define CACC0 0x01000000 /* Bit 24: Compare Accumulation Bit 0 */
#define CACC1 0x02000000 /* Bit 25: Compare Accumulation Bit 1 */
#define CACC2 0x04000000 /* Bit 26: Compare Accumulation Bit 2 */
#define CACC3 0x08000000 /* Bit 27: Compare Accumulation Bit 3 */
#define CACC4 0x10000000 /* Bit 28: Compare Accumulation Bit 4 */
#define CACC5 0x20000000 /* Bit 29: Compare Accumulation Bit 5 */
#define CACC6 0x40000000 /* Bit 30: Compare Accumulation Bit 6 */
#define CACC7 0x80000000 /* Bit 31: Compare Accumulation Bit 7 */

/* STKY register */
#define AUS   0x00000001 /* Bit 0: ALU fltg-pt. underflow */
#define AVS   0x00000002 /* Bit 1: ALU fltg-pt. overflow */
#define AOS   0x00000004 /* Bit 2: ALU fixed-pt. overflow */
#define AIS   0x00000020 /* Bit 5: ALU fltg-pt. invalid operation */
#define MOS   0x00000040 /* Bit 6: Multiplier fixed-pt. overflow */
#define MVS   0x00000080 /* Bit 7: Multiplier fltg-pt. overflow */
#define MUS   0x00000100 /* Bit 8: Multiplier fltg-pt. underflow */
#define MIS   0x00000200 /* Bit 9: Multiplier fltg-pt. invalid operation */
#define CB7S  0x00020000 /* Bit 17: DAG1 circular buffer 7 overflow */
#define CB15S 0x00040000 /* Bit 18: DAG2 circular buffer 15 overflow */
#define PCFL  0x00200000 /* Bit 21: PC stack full */
#define PCEM  0x00400000 /* Bit 22: PC stack empty */
#define SSOV  0x00800000 /* Bit 23: Status stack overflow (MODE1 and ASTAT) */
#define SSEM  0x01000000 /* Bit 24: Status stack empty */
#define LSOV  0x02000000 /* Bit 25: Loop stack overflow */
#define LSEM  0x04000000 /* Bit 26: Loop stack empty */

/* IRPTL and IMASK and IMASKP registers */
#define RSTI  0x00000002 /* Bit 1: Address: 08: Reset */
#define SOVFI 0x00000008 /* Bit 3: Address: 18: Stack overflow */
#define TMZHI 0x00000010 /* Bit 4: Address: 20: Timer = 0 (high priority) */

```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 192/212
File: AmicaSoftware.doc

```
#define IRQ3I 0x00000020 /* Bit 5: Address: 28: IRQ3- asserted */
#define IRQ2I 0x00000040 /* Bit 6: Address: 30: IRQ2- asserted */
#define IRQ1I 0x00000080 /* Bit 7: Address: 38: IRQ1- asserted */
#define IRQ0I 0x00000100 /* Bit 8: Address: 40: IRQ0- asserted */
#define CB7I 0x00000800 /* Bit 11: Address: 58: Circ. buffer 7 overflow */
#define CB15I 0x00001000 /* Bit 12: Address: 60: Circ. buffer 15 overflow */
#define TMZLI 0x00004000 /* Bit 14: Address: 70: Timer = 0 (low priority) */
#define FIXI 0x00008000 /* Bit 15: Address: 78: Fixed-pt. overflow */
#define FLT0I 0x00010000 /* Bit 16: Address: 80: fltg-pt. overflow */
#define FLTUI 0x00020000 /* Bit 17: Address: 88: fltg-pt. underflow */
#define FLTII 0x00040000 /* Bit 18: Address: 90: fltg-pt. invalid */
#define SFT0I 0x01000000 /* Bit 24: Address: C0: user software int 0 */
#define SFT1I 0x02000000 /* Bit 25: Address: C8: user software int 1 */
#define SFT2I 0x04000000 /* Bit 26: Address: D0: user software int 2 */
#define SFT3I 0x08000000 /* Bit 27: Address: D8: user software int 3 */
#define SFT4I 0x10000000 /* Bit 28: Address: E0: user software int 4 */
#define SFT5I 0x20000000 /* Bit 29: Address: E8: user software int 5 */
#define SFT6I 0x40000000 /* Bit 30: Address: F0: user software int 6 */
#define SFT7I 0x80000000 /* Bit 31: Address: F8: user software int 7 */

#endif // __DEF21020_H_

/* end of file */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 193/212
File: AmicaSoftware.doc

5.3.16. defs.h

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File:  defs.h  
//  
// Version: 1.0  
// Modified: 30/05/07  
// Author:  F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes:  Loader code definitions  
//  
////////////////////////////////////  
  
#ifndef __DEFS_H_  
#define __DEFS_H_  
  
#define ser0_data      0x80000000  /* serial 0 data DM address */  
#define ser0_reg       0x80000005  /* serial 0 register DM address */  
#define ser1_data      0x80000010  /* serial 1 data DM address */  
#define ser1_reg       0x80000015  /* serial 1 register DM address */  
#define ser_rit 128 /* delay cycles: 128 for ram, 32 for flash */  
#define adc1_ch0       0x80000030  /* ADC1 channel 0 data address */  
#define adc1_ch1       0x80000031  /* ADC1 channel 1 data address */  
#define adc1_ch2       0x80000032  /* ADC1 channel 2 data address */  
#define adc1_ch3       0x80000033  /* ADC1 channel 3 data address */  
#define adc1_ch4       0x80000034  /* ADC1 channel 4 data address */  
#define adc1_ch5       0x80000035  /* ADC1 channel 5 data address */  
#define adc1_ch6       0x80000036  /* ADC1 channel 6 data address */  
#define adc1_ch7       0x80000037  /* ADC1 channel 7 data address */  
#define adc2_ch0       0x80000038  /* ADC2 channel 0 data address */  
#define adc2_ch1       0x80000039  /* ADC2 channel 1 data address */  
#define adc2_ch2       0x8000003A  /* ADC2 channel 2 data address */  
#define adc2_ch3       0x8000003B  /* ADC2 channel 3 data address */  
#define adc2_ch4       0x8000003C  /* ADC2 channel 4 data address */  
#define adc2_ch5       0x8000003D  /* ADC2 channel 5 data address */  
#define adc2_ch6       0x8000003E  /* ADC2 channel 6 data address */  
#define adc2_ch7       0x8000003F  /* ADC2 channel 7 data address */  
  
#define flg_prg0       0x008000    /* main program 1 validation */  
#define flg_prg1       0x03F000    /* main program 2 validation */  
#define flg_prg2       0x076000    /* main program 3 validation */  
#define prg1_adr       0x010000    /* main program #1 start address */  
#define prg2_adr       0x047000    /* main program #2 start address */  
#define prg3_adr       0x07E000    /* main program #3 start address */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 194/212
File: AmicaSoftware.doc

```
#define prg_addr      0x407000    /* main program init address */  
#define mprg_len     0x02F000    /* main program length */  
#define prgstart 0x407100    /* main program execution start address */  
  
#endif // DEFS_H
```

5.3.17. imagepro.h

```
#ifndef IMAGEPROH
#define IMAGEPROH

#include <stdio.h>
#include <math.h>
#include "nrutil2.h"

typedef struct sortstruct {
    int numero;
    float x;
    float y;
    float brill;
} SORTSTAR;

char flagpsf;
int maxiter;
int colbox;
int maossim;
int passo;
int nrowssim;
int ncolssim;
double Bzero;
double Bx;
double By;
double sigma;
double sigmax;
double sigmay;
double rho;
double tau;
double fattmolt;
int nstarsim;
int addnoise;
int modpsfsim;
double exptimesim;
short xnewstar;
short ynewstar;
long maxAdu;
double magnewstar;
int maxormag;

void moments(float sigmas[],int okfitcount,float *minsig,float *maxsig,float *meansigmas,
            float *mediansigmas,float *modesigmas,float *sigmasigmas);
void sort(unsigned long n, float arr[]);
void sortstruc(unsigned long n, SORTSTAR arrstruc[]);
void fit(double x[], double y[], double sig[], int ndata, double a[],
        double iniguess[], int ia[], int ma, int *ok);
void covsrt(double **covar, int ma, int ia[], int mfit);
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 196/212
File: AmicaSoftware.doc

```
void gaussj(double **a, int n, double **b, int m, int *success);
void mrqcof(double x[], double y[], double sig[], int ndata, double a[], int ia[],
            int ma, double **alpha, double beta[], double *chisq,
            void (*funcs)(double, double [], double *, double [], int, int));
void mrqmin(double x[], double y[], double sig[], int ndata, double a[], int ia[],
            int ma, double **covar, double **alpha, double *chisq,
            void (*funcs)(double, double [], double *, double [], int, int), double *alamda,
            int *fitok);
void fgauss(double p, double a[], double *yfit, double dyda[], int ma, int col);
void fgaussub(double p, double a[], double *yfit, double dyda[], int ma, int col);
void fmoftat(double p, double a[], double *yfit, double dyda[], int ma, int col);
void fmoftatub(double p, double a[], double *yfit, double dyda[], int ma, int col);
void fgaussasimm(double p, double a[], double *yfit, double dyda[], int ma, int col);
void ossimul(double **ossim, double **sigsim,
             void (*funcs)(double, double [], double *, double [], int, int));
void parsim(double *asim);
double ranl(long *idum);
double gasdev(long *idum);
void noiseadd(double **ossim);
void addstar(double **ossim);

#endif
```

5.3.18. nrutil2.h

```

#ifndef _NR_UTILS_H_
#define _NR_UTILS_H_

static float sqrarg;
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)

static double dsqarg;
#define DSQR(a) ((dsqarg=(a)) == 0.0 ? 0.0 : dsqarg*dsqarg)

static double dmaxarg1,dmaxarg2;
#define DMAX(a,b) (dmaxarg1=(a),dmaxarg2=(b),(dmaxarg1) > (dmaxarg2)      (dmaxarg1) :
(dmaxarg2))

static double dminarg1,dminarg2;
#define DMIN(a,b) (dminarg1=(a),dminarg2=(b),(dminarg1) < (dminarg2)      (dminarg1) : (dminarg2))

static float maxarg1,maxarg2;
#define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2)      (maxarg1) : (maxarg2))

static float minarg1,minarg2;
#define FMIN(a,b) (minarg1=(a),minarg2=(b),(minarg1) < (minarg2)      (minarg1) : (minarg2))

static long lmaxarg1,lmaxarg2;
#define LMAX(a,b) (lmaxarg1=(a),lmaxarg2=(b),(lmaxarg1) > (lmaxarg2)      (lmaxarg1) : (lmaxarg2))

static long lminarg1,lminarg2;
#define LMIN(a,b) (lminarg1=(a),lminarg2=(b),(lminarg1) < (lminarg2)      (lminarg1) : (lminarg2))

static int imaxarg1,imaxarg2;
#define IMAX(a,b) (imaxarg1=(a),imaxarg2=(b),(imaxarg1) > (imaxarg2)      (imaxarg1) : (imaxarg2))

static int iminarg1,iminarg2;
#define IMIN(a,b) (iminarg1=(a),iminarg2=(b),(iminarg1) < (iminarg2)      (iminarg1) : (iminarg2))

#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

#ifdef __STDC__ || defined(ANSI) || defined(NRANSI) /* ANSI */
extern "C" {
void nrerror(char error_text[]);
float *vector(long nl, long nh);
int *ivector(long nl, long nh);
unsigned char *cvector(long nl, long nh);
unsigned long *lvector(long nl, long nh);
double *dvector(long nl, long nh);
float **matrix(long nrl, long nrh, long ncl, long nch);
double **dmatrix(long nrl, long nrh, long ncl, long nch);
int **imatrix(long nrl, long nrh, long ncl, long nch);
float **submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch,

```

```
    long newrl, long newcl);

double **dsubmatrix(double **a, long oldrl, long oldrh, long oldcl, long oldch,
    long newrl, long newcl);

float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch);
float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh);
void free_vector(float *v, long nl, long nh);
void free_ivector(int *v, long nl, long nh);
void free_cvector(unsigned char *v, long nl, long nh);
void free_lvector(unsigned long *v, long nl, long nh);
void free_dvector(double *v, long nl, long nh);
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch);
void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch);
void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch);
void free_submatrix(float **b, long nrl, long nrh, long ncl, long nch);

void free_dsubmatrix(double **b, long nrl, long nrh, long ncl, long nch);

void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch);
void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch,
    long ndl, long ndh);
}

#ifdef KR

void nrrror();
float *vector();
float **matrix();
float **submatrix();
double **dsubmatrix();
float **convert_matrix();
float ***f3tensor();
double *dvector();
double **dmatrix();
int *ivector();
int **imatrix();
unsigned char *cvector();
unsigned long *lvector();
void free_vector();
void free_dvector();
void free_ivector();
void free_cvector();
void free_lvector();
void free_matrix();
void free_submatrix();
void free_dsubmatrix();
void free_convert_matrix();
void free_dmatrix();
void free_imatrix();
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 199/212
File: AmicaSoftware.doc

```
void free_f3tensor();  
  
#endif  
  
#endif /* _NR_UTILS_H_ */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 200/212
File: AmicaSoftware.doc

5.3.19. psfintegral.h

```
#ifndef PSFINT
#define PSFINT

double gaussintegral(double max, double sigmax, double sigmay);
double quad2d(double (*func)(double, double, double, double, double),
              double x1, double x2, double max, double sigmax, double sigmay);
double f1(double x, double max, double sigmax, double sigmay);
double f2(double y, double max, double sigmax, double sigmay);
double yy1(double x, double sigmay);
double yy2(double x, double sigmay);
double qgaus(double (*func)(double, double, double, double), double a, double b,
             double max, double sigmax, double sigmay);
void gauleg(double est1, double est2, double x[], double w[], int n);
double funcgauss(double x, double y, double max, double sigmax, double sigmay);

#endif
```

5.3.20. signal.h

```
/*
 *
 * signal.h
 *
 * (c) Copyright 2001-2004 Analog Devices, Inc. All rights reserved.
 * $Revision: 1.2 $
 */

#ifndef __SIGNAL_DEFINED
#define __SIGNAL_DEFINED

#if defined(__ADSP21020__)

# define SIG_SOVF      3
# define SIG_TMZ0      4
# define SIG_IRQ3      5
# define SIG_IRQ2      6
# define SIG_IRQ1      7
# define SIG_IRQ0      8
# define SIG_CB7       11
# define SIG_CB15      12
# define SIG_TMZ       14
# define SIG_FIX       15
# define SIG_FLTO      16
# define SIG_FLTU      17
# define SIG_FLTI      18
# define SIG_USR0      24
# define SIG_USR1      25
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 201/212
File: AmicaSoftware.doc

```
# define SIG_USR2      26
# define SIG_USR3      27
# define SIG_USR4      28
# define SIG_USR5      29
# define SIG_USR6      30
# define SIG_USR7      31

# define __SIG_FIRST_INTERRUPT SIG_SOVF
# define __SIG_LAST_HARDWARE_INT SIG_USR7

#endif /* __ADSP21020__ */

/*****/

#if defined(__2106x__)

# define SIG_SOVF      3
# define SIG_TMZ0      4
# define SIG_VIRPTI    5
# define SIG_IRQ2      6
# define SIG_IRQ1      7
# define SIG_IRQ0      8
# define SIG_SPR0I     10
# define SIG_SPR1I     11
# define SIG_SPT0I     12
# define SIG_SPT1I     13

# if !defined(__ADSP21061__) && !defined(__ADSP21065L__)
# define SIG_LP2I      14
# define SIG_LP3I      15
# endif

# define SIG_EP0I      16
# define SIG_EP1I      17

# if !defined(__ADSP21061__) && !defined(__ADSP21065L__)
# define SIG_EP2I      18
# define SIG_EP3I      19
# define SIG_LSRQ      20
# endif

# define SIG_CB7        21
# define SIG_CB15       22
# define SIG_TMZ        23
# define SIG_FIX        24
# define SIG_FLTO       25
# define SIG_FLTU       26
# define SIG_FLTI       27
# define SIG_USR0       28
# define SIG_USR1       29
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 202/212
File: AmicaSoftware.doc

```
# define SIG_USR2    30
# define SIG_USR3    31

# define __SIG_FIRST_INTERRUPT SIG_SOVF
# define __SIG_LAST_HARDWARE_INT SIG_USR3

#endif /* __2106x__ */

/*****/

#if defined(__2116x__)

# define SIG_IICDI    2
# define SIG_SOVF    3
# define SIG_TMZ0    4
# define SIG_VIRPTI    5
# define SIG_IRQ2    6
# define SIG_IRQ1    7
# define SIG_IRQ0    8

# if defined(__ADSP21160__)
# define SIG_SPR0I    10
# define SIG_SPR1I    11
# define SIG_SPT0I    12
# define SIG_SPT1I    13
# else
# define SIG_SP0I    10
# define SIG_SP1I    11
# define SIG_SP2I    12
# define SIG_SP3I    13
# endif

# define SIG_LP0I    14
# define SIG_LP1I    15

# if defined(__ADSP21160__)
# define SIG_LP2I    16
# define SIG_LP3I    17
# define SIG_LP4I    18
# define SIG_LP5I    19
# define SIG_EP0I    20
# define SIG_EP1I    21
# define SIG_EP2I    22
# define SIG_EP3I    23
# define SIG_LSRQ    24
# define SIG_CB7    25
# define SIG_CB15    26
# define SIG_TMZ    27
# define SIG_FIX    28
# define SIG_FLTO    29
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 203/212
File: AmicaSoftware.doc

```
# define SIG_FLTU    30
# define SIG_FLTI    31
# define SIG_USR0    32
# define SIG_USR1    33
# define SIG_USR2    34
# define SIG_USR3    35
# else
# define SIG_SPIRI   16
# define SIG_SPITI   17
# define SIG_EP0I    18
# define SIG_EP1I    19
# define SIG_EP2I    20
# define SIG_EP3I    21
# define SIG_LSRQ    22
# define SIG_CB7     23
# define SIG_CB15    24
# define SIG_TMZ     25
# define SIG_FIX     26
# define SIG_FLTO    27
# define SIG_FLTU    28
# define SIG_FLTI    29
# define SIG_USR0    30
# define SIG_USR1    31
# define SIG_USR2    32
# define SIG_USR3    33
# endif /* ! __ADSP21161__ */

# define __SIG_FIRST_INTERRUPT SIG_IICDI
# define __SIG_LAST_HARDWARE_INT SIG_USR3

# endif /* __2116x__ */

/*****/

#if defined(__2126x__)
# define SIG_IICDI    2
# define SIG_SOVF     3
# define SIG_TMZ0     4

# define SIG_BKP      6

# define SIG_IRQ2     8
# define SIG_IRQ1     9
# define SIG_IRQ0    10
# define SIG_DAIH    11
# define SIG_SPIH    12
# define SIG_GPTMR0  13
# define SIG_SP1     14
# define SIG_SP3     15
#endif defined(__ADSP21262__) || defined(__ADSP21266__)
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 204/212
File: AmicaSoftware.doc

```
# define SIG_SP5      16
#endif
# define SIG_SP0      17
# define SIG_SP2      18
#if defined(__ADSP21262__) || defined(__ADSP21266__)
# define SIG_SP4      19
#endif
# define SIG_PP        20
# define SIG_GPTMR1   21

# define SIG_DAIL      23

# define SIG_GPTMR2   28
# define SIG_SPIL      29
# define SIG_CB7        30
# define SIG_CB15       31
# define SIG_TMZ        32
# define SIG_FIX        33
# define SIG_FLTO       34
# define SIG_FLTU       35
# define SIG_FLTI       36
# define SIG_EMUL       37
# define SIG_USR0       38
# define SIG_USR1       39
# define SIG_USR2       40
# define SIG_USR3       41

# define __SIG_FIRST_INTERRUPT SIG_IICDI
# define __SIG_LAST_HARDWARE_INT SIG_USR3
#endif

/*****/

#if defined(__213xx__)
# define SIG_IICDI      2    // IRPTL
# define SIG_SOVF       3    // IRPTL
# define SIG_TMZ0       4    // IRPTL

# define SIG_BKP        6    // IRPTL

# define SIG_IRQ2       8    // IRPTL
# define SIG_IRQ1       9    // IRPTL
# define SIG_IRQ0      10    // IRPTL
# define SIG_P0        11    // IRPTL - DAIH
# define SIG_P1        12    // IRPTL - SPIH
# define SIG_P2        13    // IRPTL - GPTMR0
# define SIG_P3        14    // IRPTL - SP1
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 205/212
File: AmicaSoftware.doc

```
# define SIG_P4      15  // IRPTL - SP3
# define SIG_P5      16  // IRPTL - SP5
# define SIG_P6      17  // LIRPTL - SP0
# define SIG_P7      18  // LIRPTL - SP2
# define SIG_P8      19  // LIRPTL - SP4
# define SIG_P9      20  // LIRPTL - PP
# define SIG_P10     21  // LIRPTL - GPTMR1
#if defined(__ADSP21367__) || defined(__ADSP21368__) || defined(__ADSP21369__) ||
defined(__2137x__)
# define SIG_P11     22  // LIRPTL - SP07
# define SIG_P12     23  // LIRPTL - DAIL
# define SIG_P13     24  // LIRPTL - PWM
# define SIG_P14     25  // IRPTL - DPII
# define SIG_P15     26  // IRPTL - DTCP
# define SIG_P16     27  //LIRPTL- SP06
#else
# define SIG_P12     23  // LIRPTL - DAIL
# define SIG_P13     24  // LIRPTL - PWM
// IRPTL
# define SIG_P15     26  // IRPTL - DTCP
// IRPTL
#endif
# define SIG_P17     28  // LIRPTL - GPTMR2
# define SIG_P18     29  // LIRPTL - SPIL
# define SIG_CB7     30  // IRPTL
# define SIG_CB15    31  // IRPTL
# define SIG_TMZ     32  // IRPTL
# define SIG_FIX     33  // IRPTL
# define SIG_FLTO    34  // IRPTL
# define SIG_FLTU    35  // IRPTL
# define SIG_FLTI    36  // IRPTL
# define SIG_EMUL    37  // IRPTL
# define SIG_USR0    38  // IRPTL
# define SIG_USR1    39  // IRPTL
# define SIG_USR2    40  // IRPTL
# define SIG_USR3    41  // IRPTL

# define SIG_DAIH    11
# define SIG_SPIH    12
# define SIG_GPTMR0  13
# define SIG_SP1     14
# define SIG_SP3     15
# define SIG_SP5     16
# define SIG_SP0     17
# define SIG_SP2     18
# define SIG_SP4     19
#if defined(__ADSP21367__) || defined(__ADSP21368__) || defined(__ADSP21369__)
# define SIG_EP0     20 // External Port0 interrupt - Corresponds to Sig_P9
# define SIG_GPTMR1  21
# define SIG_SP7     22 // SPORT 7 interrupt - Corresponds to Sig_P11 in LX3
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 206/212
File: AmicaSoftware.doc

```
# define SIG_DAIL      23
# define SIG_EP1      24 // External Port1 interrupt - Corresponds to Sig_P13
# define SIG_DPI      25 // DPII interrupt -Corresponds to Sig_P14 in LX3
# define SIG_MTM      26 // Mem to Mem Interrupt - Corresponds to Sig_P15
# define SIG_SP6      27 // SPORT 6 interrupt -Corresponds to Sig_P16 in LX3
# define SIG_GPTMR2   28
# define SIG_SPIL     29
#else
# define SIG_PP       20
# define SIG_GPTMR1   21
# define SIG_DAIL     23
# define SIG_PWM      24
# define SIG_DTCP     26
# define SIG_GPTMR2   28
# define SIG_SPIL     29
#endif

# define __SIG_FIRST_INTERRUPT SIG_IICDI
# define __SIG_LAST_HARDWARE_INT SIG_USR3

#endif

/*****/
#define __SIG_LAST_INTERRUPT __SIG_LAST_HARDWARE_INT+7

#define SIGABRT (__SIG_LAST_HARDWARE_INT+1)
#define SIGILL (__SIG_LAST_HARDWARE_INT+2)
#define SIGINT (__SIG_LAST_HARDWARE_INT+3)
#define SIGSEGV (__SIG_LAST_HARDWARE_INT+4)
#define SIGTERM (__SIG_LAST_HARDWARE_INT+5)
#define SIGALRM SIG_TMZ /* Software signal raised by timer */
#define SIGFPE (__SIG_LAST_HARDWARE_INT+7)

#define SIG_DFL ((void (*)(int))0x01)
#define SIG_ERR ((void (*)(int))0x02)
#define SIG_IGN ((void (*)(int))0x03)

#if defined(__ECC__)
/* Hide these definitions from assembly-language users in runtime support */

typedef int sig_atomic_t;

# if defined(__cplusplus)
extern "C" {
# endif

void (*signal(int _sig, void (*func)(int))) (int);
void (*sigalrm(int _sig, void (*func)(int))) (int);
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 207/212
File: AmicaSoftware.doc

```
void (*signals(int _sig, void (*func)(int))) (int);
void (*signalcb(int _sig, void (*func)(int))) (int);
void (*signalss(int _sig, void (*func)(int))) (int);

void (*signalnsm(int _sig, void (*func)(int))) (int);
void (*signalfnsm(int _sig, void (*func)(int))) (int);
void (*signalsnsm(int _sig, void (*func)(int))) (int);
void (*signalcbnsm(int _sig, void (*func)(int))) (int);
void (*signalssnsm(int _sig, void (*func)(int))) (int);

void (*interrupt(int _sig, void (*func)(int))) (int);
void (*interruptf(int _sig, void (*func)(int))) (int);
void (*interrupts(int _sig, void (*func)(int))) (int);
void (*interruptcb(int _sig, void (*func)(int))) (int);
void (*interruptss(int _sig, void (*func)(int))) (int);

void (*interruptnsm(int _sig, void (*func)(int))) (int);
void (*interruptfnsm(int _sig, void (*func)(int))) (int);
void (*interruptsnsm(int _sig, void (*func)(int))) (int);
void (*interruptcbnsm(int _sig, void (*func)(int))) (int);
void (*interruptssnsm(int _sig, void (*func)(int))) (int);

int raise(int);
int raisensm(int);

int clear_interrupt(int _sig);

# if defined(__cplusplus)
}
# endif

#endif /* __ECC__ */

#endif /* _SIGNAL_DEFINED_ */
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 208/212
File: AmicaSoftware.doc

5.3.21. star_find.h

```
////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File: star_find.h  
//  
// Version: 1.0  
// Modified: 04/06/07  
// Author: F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes: star search header file  
//  
//  
////////////////////////////////////  
  
////////////////////////////////////  
// Header files  
//  
////////////////////////////////////  
// #include <sys\exception.h>  
#include <stdio.h>  
#include <io.h>  
#include <math.h>  
#include <stdlib.h>  
#include <string.h>  
#include <signal.h>  
  
////////////////////////////////////  
// Symbolic constants  
//  
////////////////////////////////////  
#define ncol 512  
#define nrow 512  
#define maxpic 31680  
#define minsky 20  
#define maxap 12  
#define PI 3.14159  
#define VC 1.1636364  
#define scalx 1.55375  
  
////////////////////////////////////  
// Global variables  
//  
////////////////////////////////////  
unsigned int maxbox = 13;
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 209/212
File: AmicaSoftware.doc

```
unsigned int maxsky = 500;

typedef struct magni
{
    double apmag;
    float magerr;
    char name[15];
    float FIL,EFIL;
}MAG_APER;

/*
typedef struct magni
{
    double apmag;
    float magerr;
    char name[9];
    float B;
    float V;
    float R;
    float I;
}MAG_APER;
*/
bool bCloseToBorder;
float *g,*s,*h,*d;
int *hhh,*skip;
float centro_x[50],centro_y[50];
float opt[11];
float ap_r[12];
float in_sky_r;
float out_sky_r;
float tempo, JD, TU, AirMass;
float lobad,hibad,thresh,phpadu,readns;
unsigned int nstar=0;
char titolo[54],nomefile[30];
char filtro, data[11];
unsigned _stklen = 128000U;
MAG_APER *stella;

////////////////////////////////////
// Prototypes
//
////////////////////////////////////
bool InitParams( void );
bool FullSearch( void );
bool LocalSearch( void );
bool TrackStars( void );
bool InitTrackFunction( void );
void DaoMax( int iFunc );
int report(char msg[]);
int maxfind( unsigned int maxbox, unsigned int maxsky);
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 210/212
File: AmicaSoftware.doc

```
int photsb(int maxsky);  
int fsky(unsigned int max, float *smod);  
int mmm(float *sky,unsigned int nsky,float *skymn,float *skymed,float *skymod,float *sigma,float *skew);  
int load(char *nomefile, int colpar, int flag);  
void start(void);  
void centro(int x, int y, int h);  
bool CheckFOV(void);  
void seeing(float xcen, float ycen, float sky, float *FWHM, float *sigma);  
int rout1(const float *alfa, const float *beta);  
float amin1(float x,float y);  
float amax1(float x,float y);  
int sign1(float x);
```

5.3.22. utils.h

```
//////////////////////////////////////  
//  
// AMICA FOR AMS  
//  
// File:  utils.h  
//  
// Version: 1.0  
// Modified: 02/03/07  
// Author:  F.Roncella  
// Hardware: ADSP-21020  
// Project: StarTracker  
//  
// Notes:  various utility functions (header file)  
//  
//  
//////////////////////////////////////  
  
//////////////////////////////////////  
// Header files  
//  
//////////////////////////////////////  
// #include <sys\exception.h>  
// #include <cdefBF537.h>  
  
//////////////////////////////////////  
// Symbolic constants  
//  
//////////////////////////////////////  
  
//////////////////////////////////////  
// Global variables  
//  
//////////////////////////////////////  
//extern short sLight_Move_Direction;  
//extern unsigned short ucActive_LED;  
int iTestCount;  
  
//////////////////////////////////////  
// Prototypes  
//  
//////////////////////////////////////  
// in file utils.c  
void test( void );  
void SommaImmagini(double **frameA, double **frameB, double **frameSum,  
                   int rigframe, int colframe);  
void DifferenzaImmagini(double **frameA, double **frameB, double **frameDiff,  
                        int rigframe, int colframe);
```



AMICA FOR AMS

SOFTWARE DESCRIPTION

DOC.: AMIA/ SWDS /1/A
ISSUE: 1
DATE: Jun 07
PAGE: 212/212
File: AmicaSoftware.doc

```
void ImmaginePerScalare(double **frameOrig, double **frameNew, double fattmol,  
                        int rigframe, int colframe);  
void ImmaginePiuScalare(double **frameOrig, double **frameNew, double fattadd,  
                        int rigframe, int colframe);
```